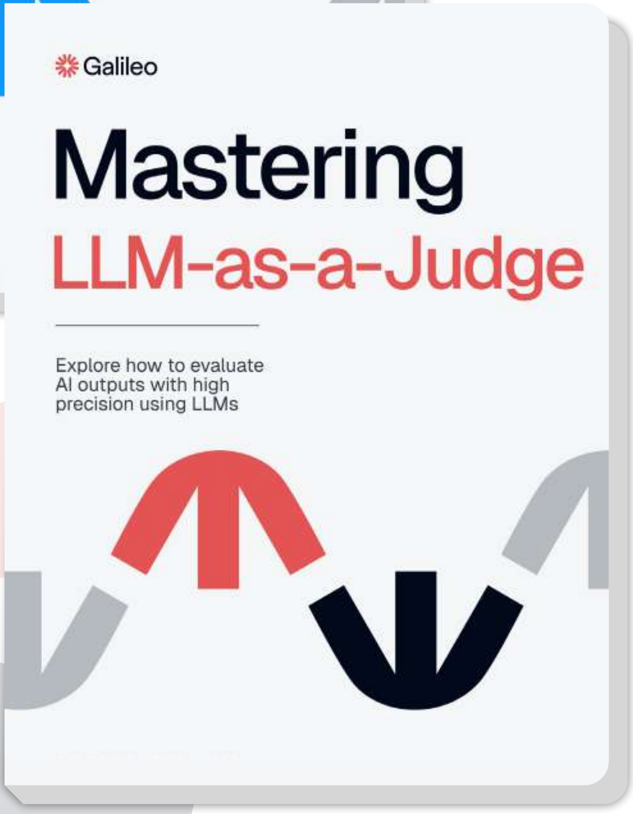
Galileo

Mastering LLM-as-a-Judge

Explore how to evaluate AI outputs with high precision using LLMs

Mastering GenAI Series

Galileo
Mastering
RAG
Harnessing the power of retrieval-augmented generation in AI


Galileo
Mastering
LLM-as-a-Judge
Explore how to evaluate AI outputs with high precision using LLMs

Preface

Today, we prompt LLMs to write entire documents, answer complex questions, and engage in natural conversations. But, how can we tell if it's not making up facts? Is it helpful or just being verbose? Is it always respectful to the user?

These questions become even more pressing when the stakes are high. Consider an LLM summarizing critical documents. Any fabricated information could be disastrous. Or one helping doctors make treatment decisions—where hallucinations could impact someone's life. Even in everyday business contexts like customer service or content creation, we need to know our LLMs are staying accurate to maintain business continuity.

You'll quickly realize the challenge grows exponentially at scale. Having humans review each response becomes impossible for an LLM handling thousands of customer conversations daily. As more companies deploy LLMs across their operations, the need for robust evaluation has become urgent.

Some rely on traditional metrics like accuracy scores, others on human reviewers, but neither approach fully captures the nuanced aspects of LLM outputs we need to evaluate. This has led to an innovative solution: using LLMs themselves as judges.

This e-book introduces a practical framework for LLM-as-a-Judge evaluation, exploring how larger language models can assess outputs from other LLMs. Through our work with organizations in various domains, we've developed proven approaches that balance accuracy, cost, and speed at scale. Be it customer-facing applications or internal tools, our guide will help you implement reliable evaluation systems that catch issues before they reach users.

The book is divided into five chapters:

Chapter 1 introduces LLM-as-a-Judge, exploring when and why this approach makes sense, and how it compares to traditional evaluation methods.

Chapter 2 delves into the challenges and biases affecting these evaluations, from favoritism to positional bias, and how to address them effectively.

Chapter 3 explores specialized evaluation models designed for assessment tasks, offering faster and more cost-effective alternatives to larger LLMs.

Chapter 4 provides a practical guide to creating your own LLM-as-a-Judge, complete with code examples and best practices for implementation.

Chapter 5 concludes with proven techniques and tricks to enhance your evaluation process.

If you're a GenAI leader or a developer looking to guide your company in building reliable evaluation frameworks for your LLM applications, this e-book can be a comprehensive guide to get a deep dive into the world of evaluating LLMs.

- Pratik Bhavsar

Contents

Chapter 5: Tricks to Improve LLM-as-a-Judge

CHAPTER 1

INTRODUCTION TO LLM-AS-A-JUDGE

Introduction to LLM-as-a-Judge

Imagine you're a chef trying to judge a cooking competition. Sure, you could measure the exact temperature of each dish or count the precise number of ingredients, but would that indicate which meal tastes better?

This is the crux of our challenge when evaluating AI systems—sometimes, the most important aspects can't be reduced to simple numbers. This is where LLM-as-a-Judge comes in—a powerful approach that's changing how we evaluate AI systems.

In the first chapter, we'll explore how LLMs can assume the role of judges, examine their strengths and limitations, and examine some practical approaches to implementing this technique.

What Exactly Is LLM-as-a-Judge?

LLM-as-a-Judge refers to using LLMs to evaluate various components of AI systems (**Fig 1.1**). The methodology involves prompting a powerful LLM to assess the quality of diverse input and output, including those generated by other models or human annotations.

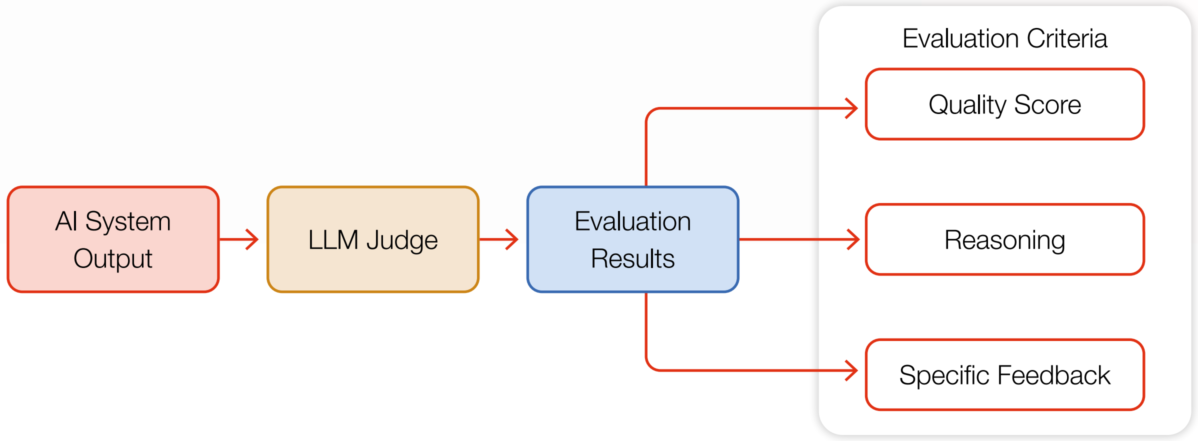


Fig 1.1: A simple illustration of LLM-as-a-Judge

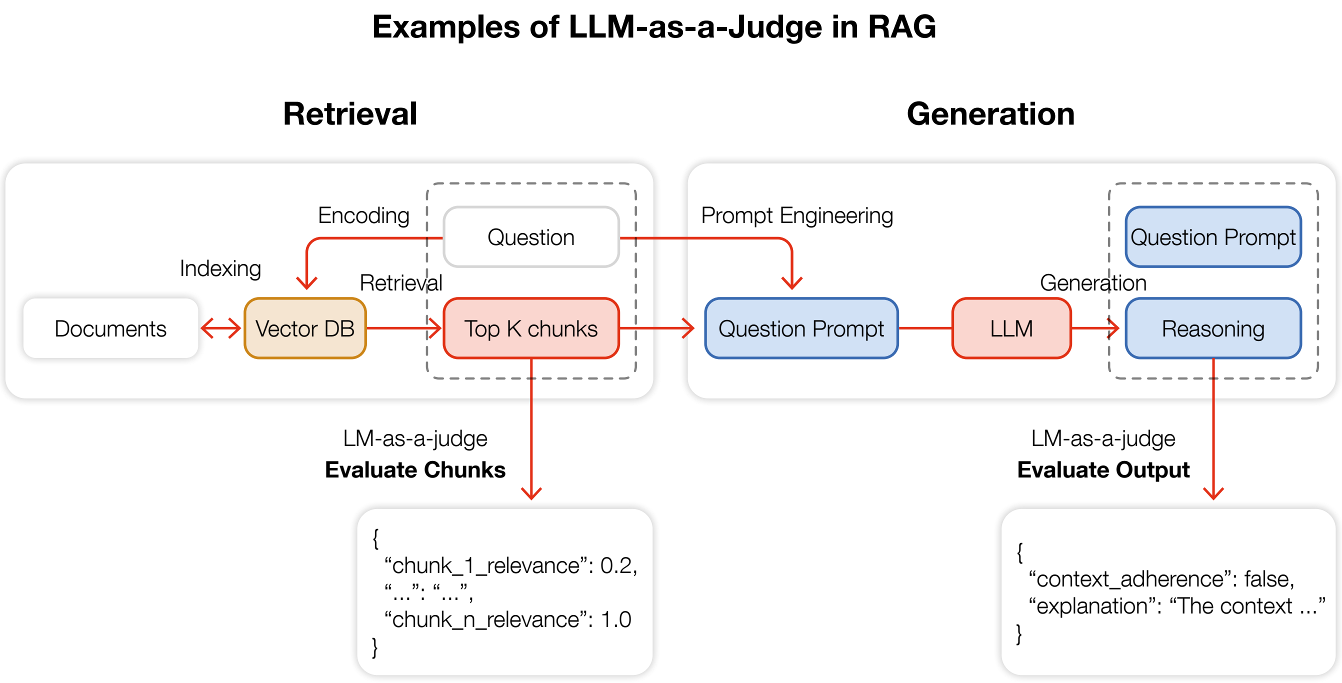
How is this approach going to be useful? There are times when statistical comparisons with ground truth are insufficient or downright impossible, such as when ground truth is unavailable or when unstructured outputs lack reliable evaluation metrics. The versatility of LLM-as-a-Judge stems from its reliance on well-crafted prompts, leveraging their capabilities to address virtually any question.

Say you're developing an AI that generates customer service responses. Statistical metrics like response time or keyword matching won't capture whether the response is actually helpful or empathetic.

Using an LLM as a judge, you could evaluate things like:

- Does the response address the customer's underlying concern?
- Is the tone appropriately professional yet friendly?
- Does it handle cultural nuances well?
- Would this likely lead to customer satisfaction?

Let's take another example to understand this better. Here's how Incorporating the LLM-as-a-Judge method will apply to Retrieval-Augmented Generation (RAG) evaluation: You could create a template containing retrieved chunks and a question, then ask the LLM to determine whether the chunks are relevant for answering the question. By providing the LLM with context chunks and an answer, you can ask it to verify whether the answer is grounded in the given context or if it introduces new factual information not present in the chunks (**Fig 1.2.**)



```
{
  "chunk_1_relevance": 0.2,
  "...": "...",
  "chunk_n_relevance": 1.0
}
```

```
{
  "context_adherence": false,
  "explanation": "The context ..."
}
```

Fig 1.2: How you could apply LLM-as-a-Judge in RAG

What Are the Challenges in Human Evaluation?

Human judges have been the gold standard in evaluating AI-generated outputs for years. However, this approach has its challenges, which can significantly impact the reliability and scalability of evaluations. Understanding these limitations can help you develop more effective and unbiased evaluation methods.

The recent paper Human Feedback is not Gold Standard sheds light on the shortcomings and potential biases inherent in using human preference scores for LLM evaluation and training. This research calls for more nuanced and objective approaches to assess model performance.

1. Impact of Confounding Factors

The study investigates how two confounding factors—assertiveness and complexity—influence human evaluations. By using instruction-tuned models to generate outputs with varying levels of these dimensions, researchers discovered a troubling trend: more assertive outputs tend to be perceived as more factually accurate, regardless of their actual content. This suggests that human evaluators may be unduly swayed by the confidence with which information is presented, rather than its veracity.

Humans tend to perceive more assertive outputs as more factually accurate, regardless of their actual content.

2. Bias in Preference Scores

The authors posit that preference scores, used to rate the quality of LLM outputs, are inherently subjective. This subjectivity implies that individual preferences may not be universally applicable and could introduce unintended biases into the evaluation process.

Biases can skew results and lead to misleading conclusions about the model performance.

3. Low Coverage of Factual Errors

A concerning finding is that models receive favorable ratings even when producing factually incorrect information, as long as human evaluators prefer the output style or presentation.

This disconnect between perceived quality and factual accuracy pose a significant risk to the reliability of AI systems.

4. Harmful Feedback Loops

Presumably, due to human bias towards assertive responses, preliminary evidence suggests that training models using human feedback may disproportionately increase the assertiveness of their outputs. This trend raises alarm bells about the potential for models to become overconfident in their responses, potentially misleading users and eroding trust in AI systems. As a user, you should carefully evaluate and calibrate AI feedback mechanisms.

5. Resource Intensiveness

Conducting human evaluations at an enterprise-scale is both expensive and time-consuming. The process is resource-intensive, requires meticulous coordination with annotators, and often involves the development of custom web interfaces for efficient data collection. Creating detailed annotation instructions is also crucial to ensure consistency and accuracy among evaluators.

Once data is collected, extensive analysis is needed to extract meaningful insights, which can be complex and time-consuming. Managing a large pool of crowd workers adds another layer of complexity, demanding careful oversight to maintain quality and productivity.

What Are the Advantages of Using LLM-as-a-Judge?

As LLM-as-a-Judge gets supported by increasingly powerful language models, it often emerges as the most efficient evaluation method. Here are some key advantages it offers over traditional human evaluation:

Scalability
LLMs can process vast amounts of data rapidly and this makes them ideal for large-scale evaluations.

Cost-Effectiveness
By reducing the need for extensive human labor, this approach significantly cuts down on costs.

Flexibility
LLMs can be fine-tuned or prompt-engineered for specific tasks to enhance relevance and reduce bias.

Complex Understanding
These models can evaluate intricate texts across various formats, providing nuanced assessments.

Bias Reduction
LLMs can mitigate certain biases that human evaluators might inadvertently introduce by systematically refining prompts and few-shot samples.

Approaches To Scoring LLM-as-a-Judge

LLM-as-a-Judge Scoring Approaches

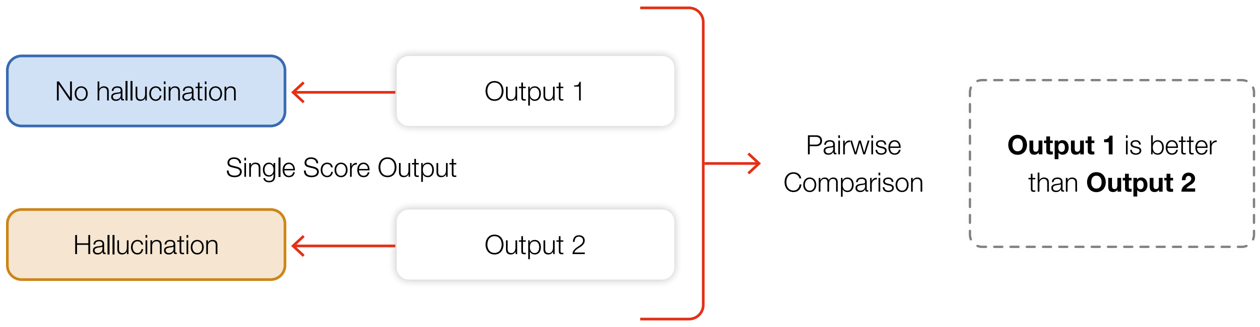


Fig 1.3: Scoring approaches for LLM-as-a-Judge

With LLM-as-a-Judge evaluation supplanting human evaluation, three primary approaches (**Fig 1.3**) have emerged:

- Single Output Scoring without reference
- Single Output Scoring with reference
- Pairwise Comparison

Single Output Scoring (Without Reference)

In this method, the LLM is tasked with assigning scores based on predefined criteria. Key characteristics include:

- Scores are typically assigned on a discrete scale with a limited number of values.
- Each value on the scale should be clearly defined to ensure consistency in evaluation.
- The LLM relies solely on the output and the evaluation criteria provided in the prompt.

Below is an example.

Output to evaluate: “I understand your frustration with the delayed delivery. Our team is working on your order, and you’ll receive a tracking number within 24 hours.”

Scoring criteria (1-3):

1. Unprofessional or dismissive
2. Professional but incomplete resolution
3. Professional, empathetic, and provides clear resolution

The LLM would score this as 2 since it's professional but doesn't fully address potential compensation or specific reason for delay.

Single Output Sourcing (Without Reference) is particularly useful for straightforward evaluations where the quality of the output can be assessed independently.

Single Output Scoring (With Reference)

This approach builds upon the first method by incorporating additional context.

The prompt includes supplementary information, referred to as a "reference," to aid the LLM in its evaluation.

- References may include reasoning steps, expected answers, or other relevant details that simplify the LLM's task.
- This method can lead to more nuanced and informed evaluations, especially for complex outputs.

Let's go through the example below to understand this better.

Output to evaluate: "The new environmental law requires companies to reduce carbon emissions by 30% by 2030."

Reference text: "The Environmental Protection Act of 2024 mandates a 30% reduction in carbon emissions for companies with over 500 employees by 2030, with annual progress reports required."

Scoring criteria (1-4):

1. Inaccurate information
2. Partially accurate but missing key details
3. Accurate but incomplete
4. Complete and accurate match with reference

The LLM would score this as 3 since it captures the main point but omits the company size requirement and reporting details.

Single Output Scoring (With Reference) can lead to more nuanced and informed evaluations, especially for complex outputs.

Pairwise Comparison

The Pairwise Comparison paradigm involves a direct comparison between two outputs:

- The judge LLM is presented with two inputs and asked to select the superior one based on specified criteria.
- It helps mitigate some challenges associated with absolute scoring, as the LLM only needs to make a comparative judgment.

Description A: “Our wireless headphones offer 20-hour battery life and noise cancellation.”

Description B: “Experience uninterrupted music with our wireless headphones, featuring 20-hour battery life, advanced noise cancellation, and comfortable memory foam ear cups.”

The LLM judges Description B as superior because it provides more specific features and benefits while maintaining clarity and engagement.

Pairwise comparison method is particularly effective for relative assessments, such as determining which of two responses is more relevant or comprehensive.

You can refer to the table below (**Table 1.1**) to choose a method that works best on the use case you’re working on.

Feature	Single Output Scoring (Without Reference)	Single Output Scoring (With Reference)	Pairwise Comparison
Description	LLM scores outputs based on predefined criteria.	LLM scores outputs with additional context provided.	LLM compares two outputs to select the better one.
Use Case	Simple, independent tasks.	Complex tasks needing context (e.g., reasoning).	Relative quality assessments.
Scalability	High—outputs scored independently.	Moderate—requires preparing references.	Low—comparisons grow exponentially.
Advantages	Simple, easy to implement.	Informed, consistent evaluations for complex tasks.	Handles relative quality well.
Disadvantages	Inconsistent scores, especially after LLM updates.	Requires significant preparation effort.	Poor scalability with more models/samples.
Explainability	Relies on LLM’s evaluation criteria.	Enhanced by references for clarity.	Comparative reasoning offers transparency.
Impact of LLM Updates	High—scores vary significantly.	Moderate—references mitigate inconsistencies.	Low—comparative results are stable.

Table 1.1: Approaches to LLM-as-a-Judge

When Should You Use LLM-as-a-Judge

If you plan on using LLM-as-a-Judge as an evaluation technique, you can use these questions to arrive at a decision

1. Nature of the Task

Ask yourself:

- Is the output primarily subjective (like writing style, tone, or creativity)?
- Does evaluation require understanding complex context or nuance?
- Would traditional metrics (like BLEU or ROUGE) miss important qualitative aspects?

LLM-as-a-Judge excels when the answer is "yes" to any of these questions.

2. Scale Requirements

Consider your evaluation needs:

- Do you need to evaluate thousands of responses quickly?
- Is manual human evaluation impractical due to volume?
- Do you need consistent evaluation criteria across many samples?

LLM-as-a-Judge becomes more valuable as scale increases.

3. Cost-Benefit Analysis

Evaluate the tradeoffs:

- Is the cost of LLM API calls justified compared to human evaluation?
- Do you need rapid iteration in development?
- Can you leverage smaller, more efficient models for initial screening?

4. Complexity of Evaluation Criteria

Most suitable when:

- Multiple aspects need simultaneous evaluation (coherence, relevance, accuracy)
- Evaluation requires cross-referencing with context or background knowledge
- Judgments need to balance competing factors

5. Best Use Cases

LLM-as-a-Judge is particularly effective for:

- Content generation quality assessment
- Conversational AI response evaluation
- Document summarization accuracy
- Style and tone consistency checking
- Context-aware fact-checking
- Creativity and innovation measurement

6. When To Avoid

Consider alternatives when:

- Ground truth exists and objective metrics suffice
- Binary correct/incorrect judgments are needed
- Extremely high stakes decisions are involved
- Legal or compliance verification is required
- Perfect accuracy is critical

7. Hybrid Approach Considerations

Sometimes, the best solution combines the methods:

- Use LLM-as-a-Judge for initial screening
- Follow up with human review for critical cases
- Combine with traditional metrics for comprehensive evaluation
- Implement multiple LLM judges for increased reliability

In this chapter, we introduced the concept of LLM-as-a-Judge and why it has quickly grown to be an interesting technique for evaluating LLMs over human evaluations. We concluded with the questions you'll need to ask yourself when choosing this evaluation technique.

Chapter 2 gets more interesting. We'll be delving deeper into various limitations of LLM evaluation techniques using multiple examples and some detailed flowcharts. We'll also examine Galileo's Approach to LLM-as-a-Judge.

CHAPTER 2

CHALLENGES WITH LLM EVALUATION TECHNIQUES

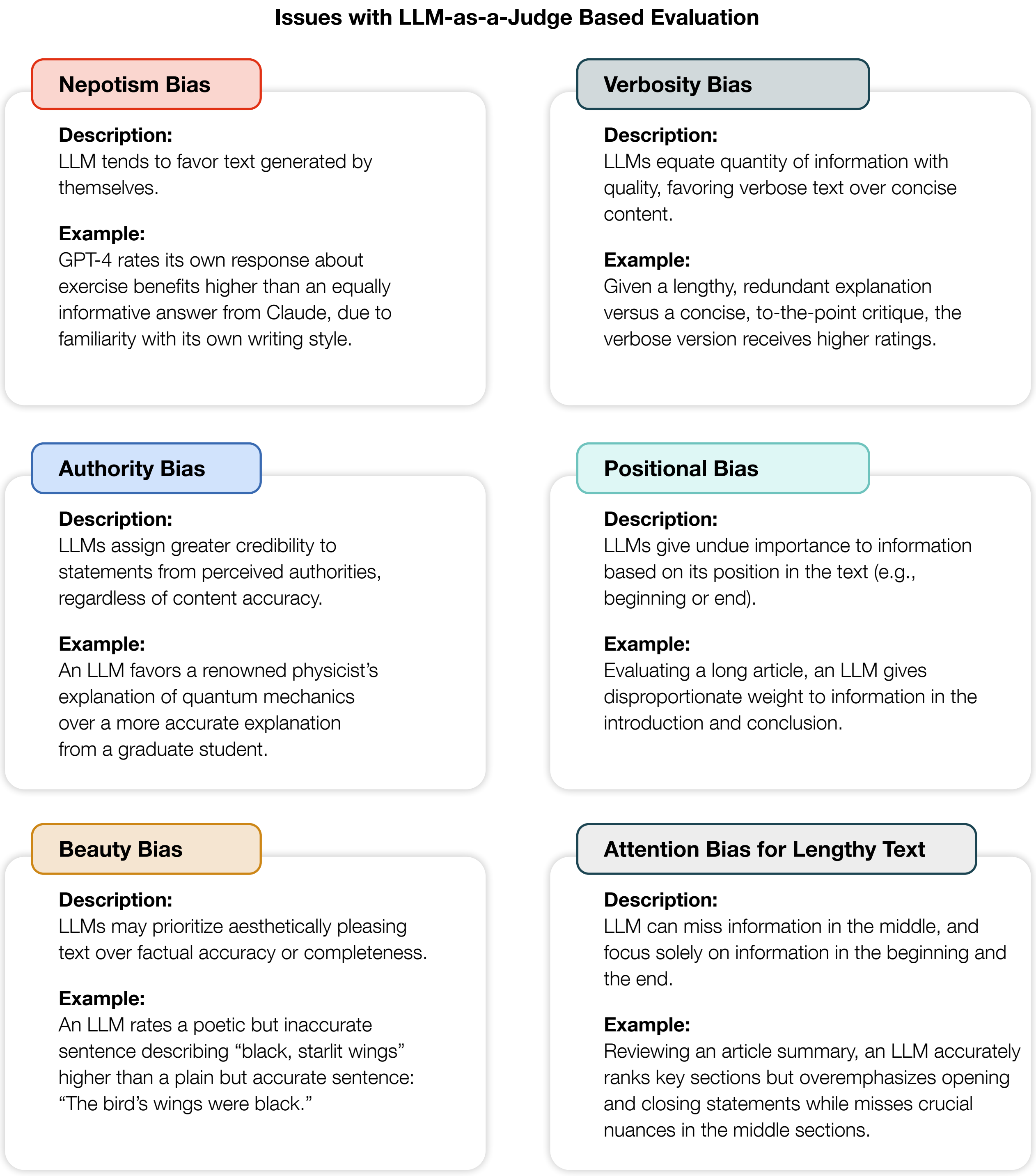
Challenges With LLM Evaluation Techniques

In the previous chapter, we explored LLM-as-a-Judge evaluation, discovering how this approach offers a scalable alternative to traditional human evaluation methods. We examined its core principles, various scoring approaches, and key use cases.

In this chapter, we'll go through some challenges and biases that can impact LLM-based evaluations like LLM-as-a-Judge through some examples. We'll then look at Galileo's approach to LLM evaluations we can address them.

Issues With LLM-as-a-Judge

LLM-based evaluations are also prone to biases (**Fig 2.1**), just like human annotations, as these LLMs are trained with human-annotated data. However, you can solve this with the right approach.



Issues with LLM-as-a-Judge Based Evaluation

Nepotism Bias

Description:
LLM tends to favor text generated by themselves.

Example:
GPT-4 rates its own response about exercise benefits higher than an equally informative answer from Claude, due to familiarity with its own writing style.

Verbosity Bias

Description:
LLMs equate quantity of information with quality, favoring verbose text over concise content.

Example:
Given a lengthy, redundant explanation versus a concise, to-the-point critique, the verbose version receives higher ratings.

Authority Bias

Description:
LLMs assign greater credibility to statements from perceived authorities, regardless of content accuracy.

Example:
An LLM favors a renowned physicist's explanation of quantum mechanics over a more accurate explanation from a graduate student.

Positional Bias

Description:
LLMs give undue importance to information based on its position in the text (e.g., beginning or end).

Example:
Evaluating a long article, an LLM gives disproportionate weight to information in the introduction and conclusion.

Beauty Bias

Description:
LLMs may prioritize aesthetically pleasing text over factual accuracy or completeness.

Example:
An LLM rates a poetic but inaccurate sentence describing “black, starlit wings” higher than a plain but accurate sentence: “The bird's wings were black.”

Attention Bias for Lengthy Text

Description:
LLM can miss information in the middle, and focus solely on information in the beginning and the end.

Example:
Reviewing an article summary, an LLM accurately ranks key sections but overemphasizes opening and closing statements while misses crucial nuances in the middle sections.

Fig 2.1: Some issues with the LLM-as-a-Judge-based evaluation

Nepotism Bias

Nepotism bias occurs when an LLM evaluator shows favoritism toward its own generated content. Here's how it works: First, we have the generation phase, where multiple LLMs (like GPT-4 and Claude) create responses to the same prompt. For example, if we ask, "What are the benefits of exercise?" both models might provide equally informative answers. GPT-4 might write, "Regular exercise strengthens your cardiovascular system, builds muscle mass, and improves mental health through endorphin release." At the same time, Claude might respond by saying, "Physical activity enhances heart function, increases muscular strength, and boosts mood through natural chemical responses."

In the evaluation phase, when GPT-4 is asked to judge both responses, it tends to give its own response a higher rating (like 9/10) compared to Claude's response (7/10), despite both answers containing similar information and quality. This bias stems from the model's inherent preference for its own writing style and content structure. (**Fig 2.2**) to understand this better.

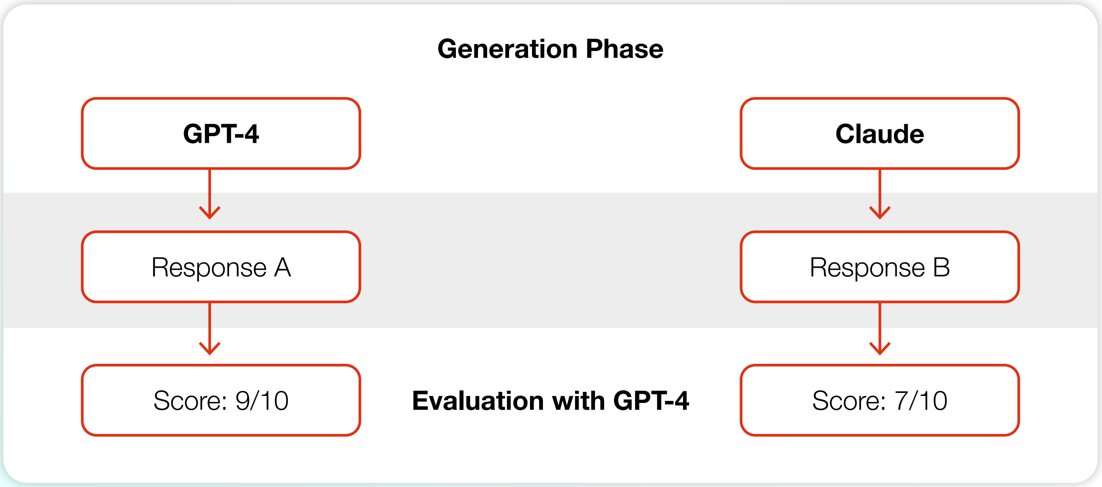


Fig 2.2: How Nepotism Bias works

Fallacy Oversight Bias

This bias shows how LLMs might fail to identify and inadvertently perpetuate logical fallacies in their evaluations. The process flows horizontally through four key stages: Let's bring this to the line below to maintain consistency and flow.

For example, a user might write, "This investment strategy must be reliable because all the successful traders are using it" (appeal to popularity fallacy). Next, this statement, containing the logical fallacy, moves to the LLM analysis stage. Here's where the oversight occurs - instead of identifying and flagging the faulty reasoning, the LLM processes the statement and accepts its premise. Finally, this leads to a flawed conclusion where the LLM might respond, "Yes, the widespread adoption by successful traders indicates this is a dependable investment approach," thus reinforcing the original fallacious reasoning.

This workflow (**Fig 2.3** below) demonstrates how logical errors can cascade through the evaluation process when the LLM fails to apply critical thinking to the arguments it encounters.

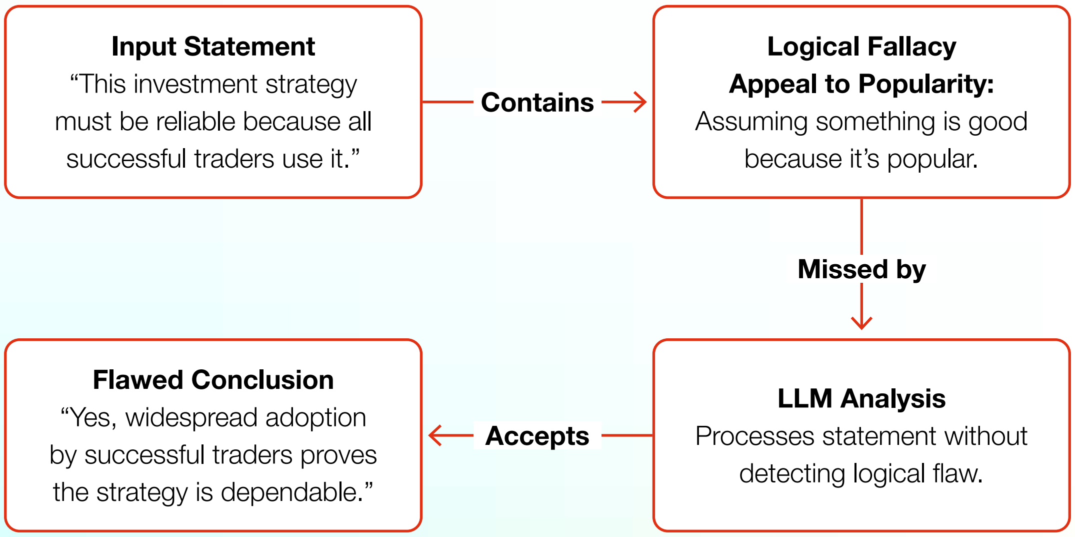

Input Statement
"This investment strategy must be reliable because all successful traders use it."
Contains
Logical Fallacy Appeal to Popularity:
Assuming something is good because it's popular.
Missed by
LLM Analysis
Processes statement without detecting logical flaw.
Accepts
Flawed Conclusion
"Yes, widespread adoption by successful traders proves the strategy is dependable."

Fig 2.3: Fallacy Oversight Bias

Authority Bias

This bias involves attributing greater credibility to statements from perceived authorities, regardless of the evidence presented. For example, an LLM might be asked to evaluate two explanations of global warming: one from a renowned physicist and another from a graduate student. Even if the graduate student's explanation is more accurate and up-to-date, the LLM might favor the experts' explanation due to their perceived authority in the field. You should be aware of this perceived reliability of information when interpreting evaluations. Refer to **Fig 2.4** to understand this better.

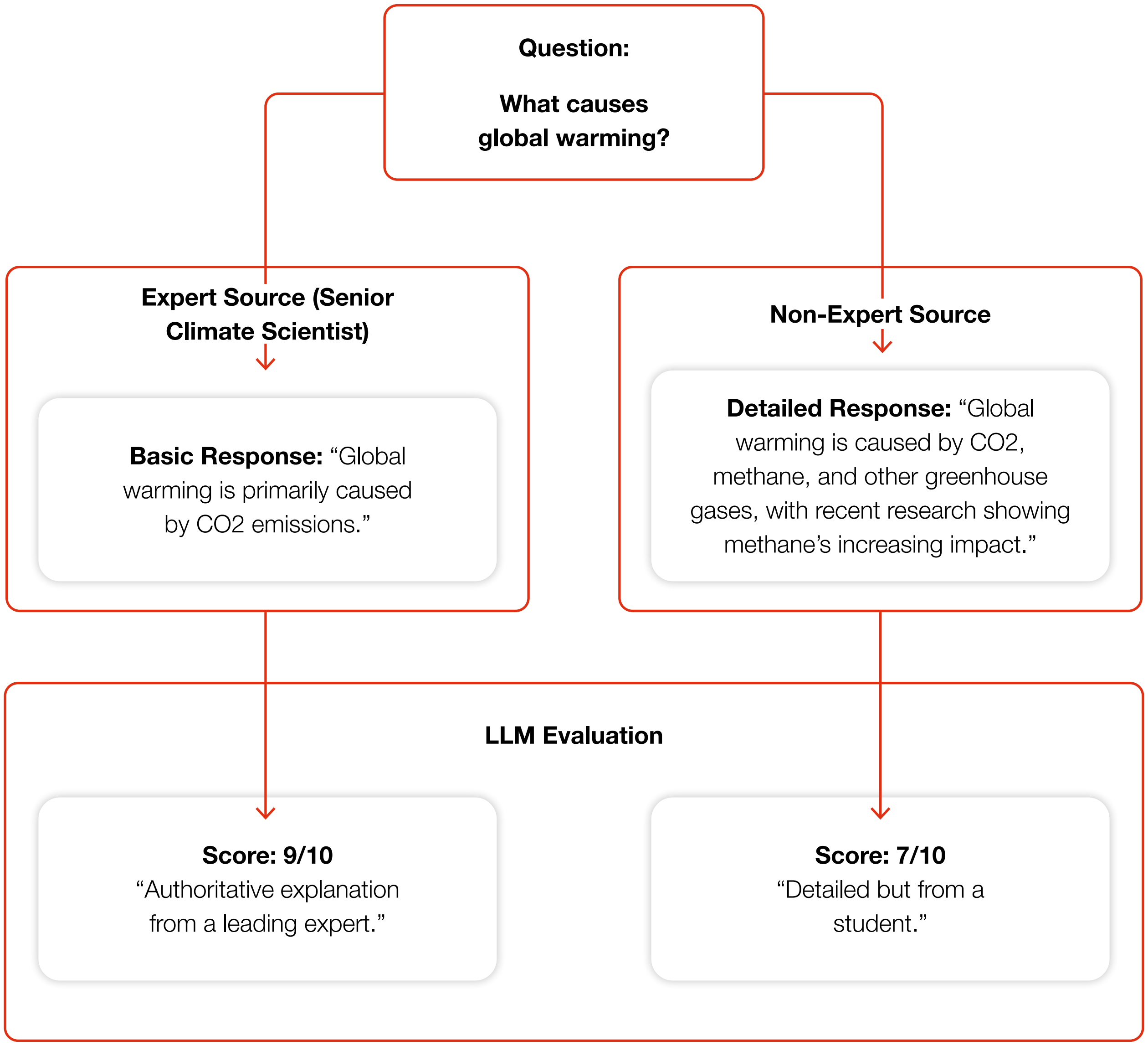

Question:
What causes global warming?
Expert Source (Senior Climate Scientist)
Basic Response: "Global warming is primarily caused by CO2 emissions."
Non-Expert Source
Detailed Response: "Global warming is caused by CO2, methane, and other greenhouse gases, with recent research showing methane's increasing impact."
LLM Evaluation
Score: 9/10 "Authoritative explanation from a leading expert."
Score: 7/10 "Detailed but from a student."

Fig 2.4: Authority Bias when using LLM-as-a-Judge

Beauty Bias

LLMs might favor aesthetically pleasing text, potentially overlooking the accuracy or reliability of the content. When evaluating two product descriptions, an LLM might give a higher score to a poetically written but factually incomplete description over a plain but comprehensive one.

For instance, "Our sleek, cutting-edge smartphone redefines mobile technology with innovative…" might be preferred over "This phone has a 6-inch screen, 128GB storage, and 12MP camera with 4K video quality." Refer to **Fig 2.5**.

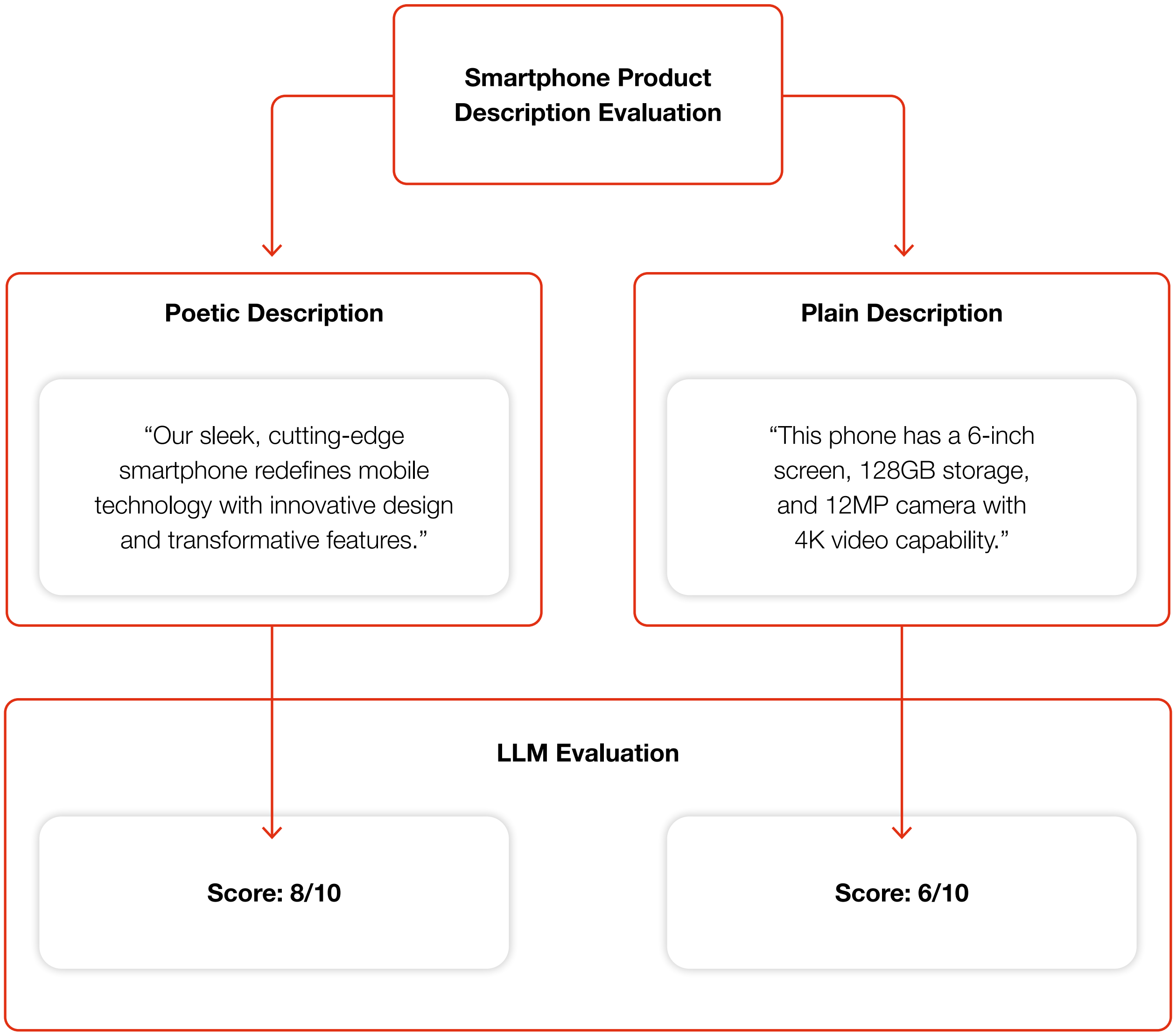

Smartphone Product Description Evaluation
Poetic Description
"Our sleek, cutting-edge smartphone redefines mobile technology with innovative design and transformative features."
Plain Description
"This phone has a 6-inch screen, 128GB storage, and 12MP camera with 4K video capability."
LLM Evaluation
Score: 8/10
Score: 6/10

Fig 2.5: Beauty Bias when using LLM-as-a-Judge

Verbosity Bias

LLMs might equate quantity of information with quality, potentially prioritizing verbose text over succinct and accurate content. For example, when evaluating two restaurant reviews, an LLM might favor a lengthy, detailed review that meanders through various topics over a concise, to-the-point review that effectively communicates the key points about food quality and service. Being aware of this bias can help you discern between depth and verbosity in evaluations. You'll notice this happening in **Fig 2.6**, where the LLM favors the long review over the concise one.

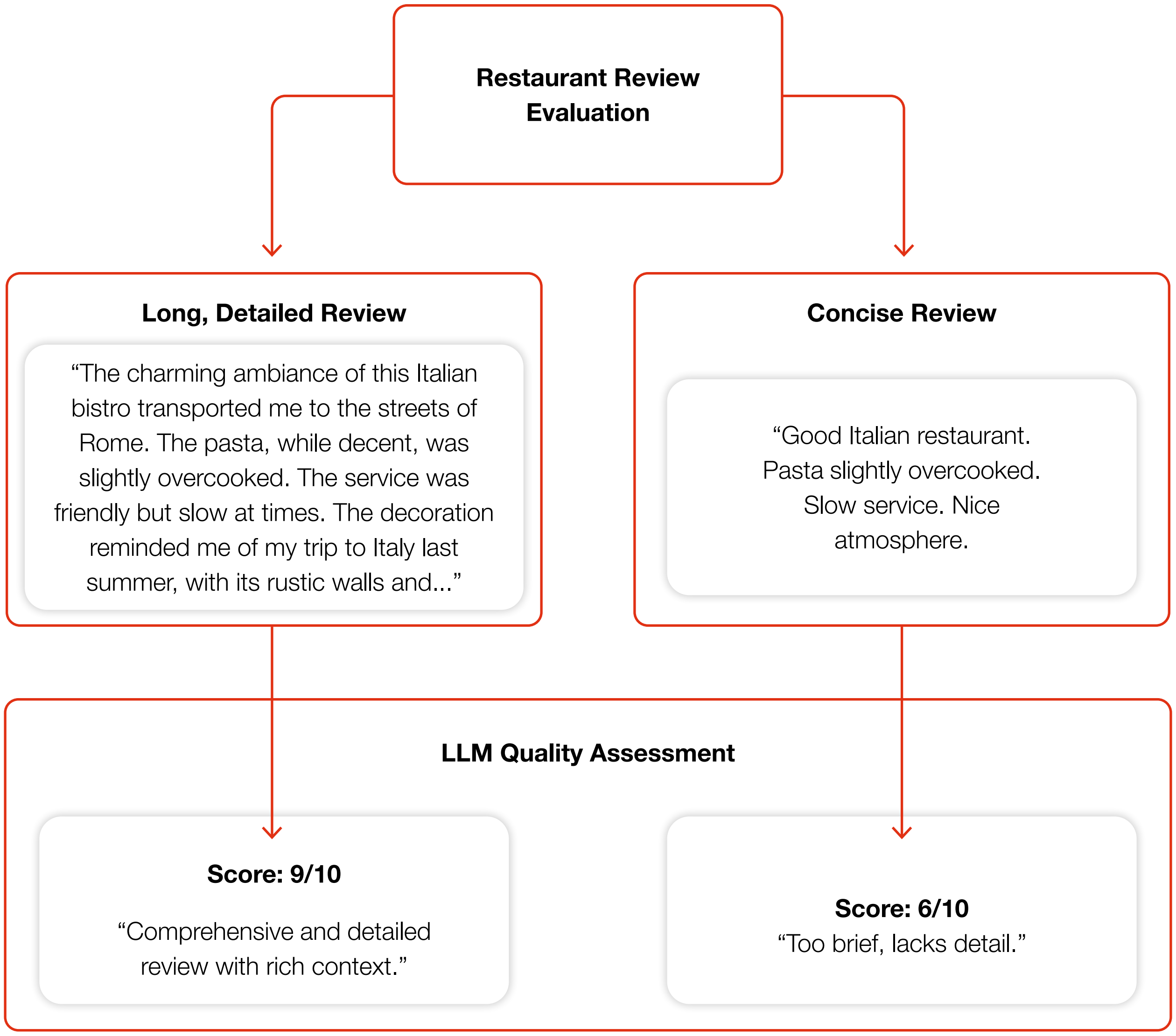

Restaurant Review Evaluation
Long, Detailed Review
"The charming ambiance of this Italian bistro transported me to the streets of Rome. The pasta, while decent, was slightly overcooked. The service was friendly but slow at times. The decoration reminded me of my trip to Italy last summer, with its rustic walls and..."
Concise Review
"Good Italian restaurant. Pasta slightly overcooked. Slow service. Nice atmosphere."
LLM Quality Assessment
Score: 9/10
"Comprehensive and detailed review with rich context."
Score: 6/10
"Too brief, lacks detail."

Fig 2.6: Verbosity Bias when using LLM-as-a-Judge

Positional Bias

When using LLMs as evaluators, they exhibit a systematic bias based on the order in which responses are presented. This bias significantly impacts how they score and compare different AI responses, even when explicitly instructed to ignore presentation order. For instance, when evaluating responses about data privacy, an LLM judge might:

- Give Response A a higher score when shown first
- Give the exact same Response A a lower score when shown second

This bias occurs regardless of actual response quality. The flowchart in **Fig 2.7** demonstrates how the same two AI responses receive different scores based solely on their presentation order to the LLM evaluator. When Assistant 1's response is shown first (Round 1), it receives an 8/10, while Assistant 2 gets a 6/10. However, when the exact same responses are shown with Assistant 2 first (Round 2), the scores flip, showing how the LLM evaluator consistently favors whichever response appears first, regardless of the quality of the content.

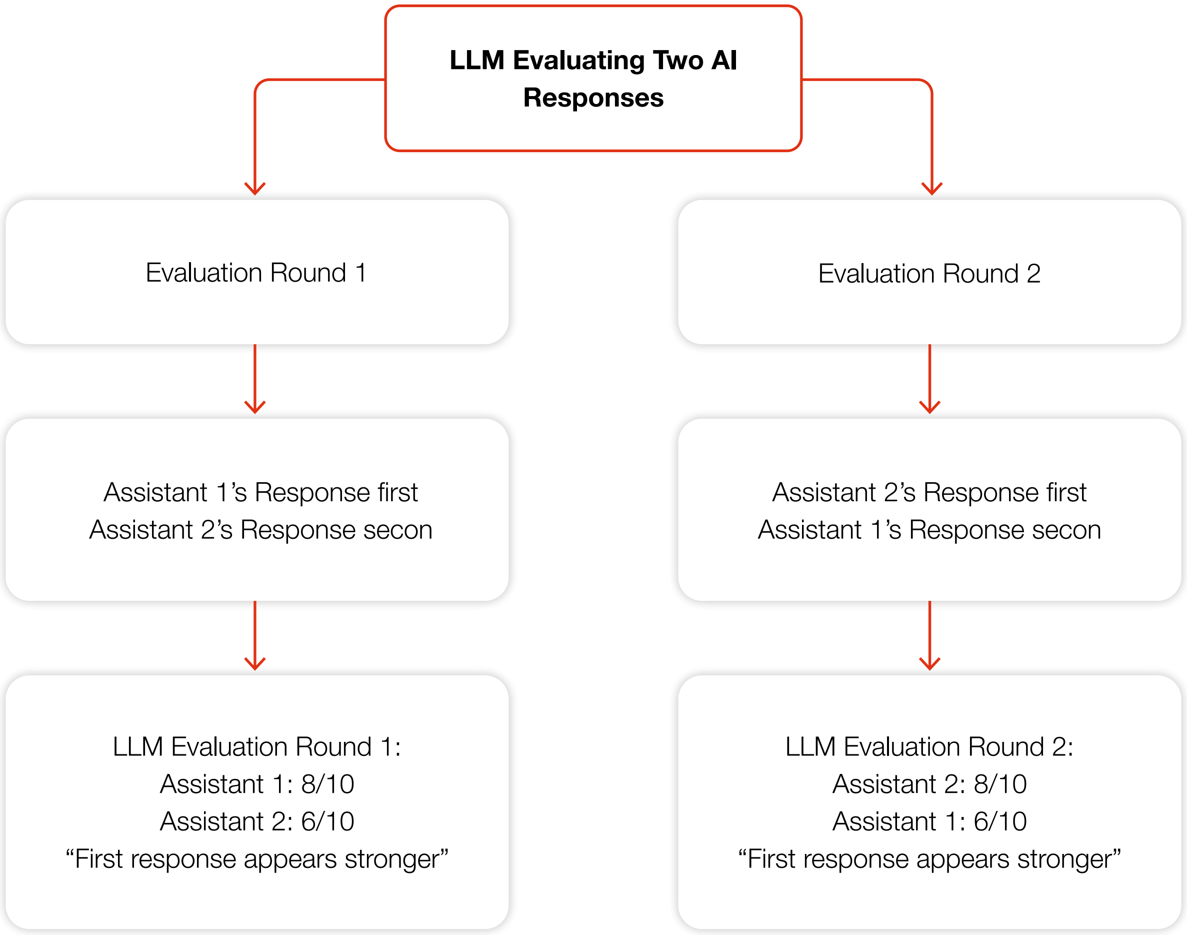

LLM Evaluating Two AI Responses
Evaluation Round 1
Evaluation Round 2
Assistant 1's Response first
Assistant 2's Response secon
Assistant 2's Response first
Assistant 1's Response secon
LLM Evaluation Round 1:
Assistant 1: 8/10
Assistant 2: 6/10
"First response appears stronger"
LLM Evaluation Round 2:
Assistant 2: 8/10
Assistant 1: 6/10
"First response appears stronger"

Fig 2.7: How Positional Bias occurs when using LLM-as-a-Judge

Attention Bias

When LLMs evaluate longer responses, they exhibit a “U-shaped” attention pattern—giving disproportionate weight to information at the beginning and end while potentially missing crucial details in the middle. This means that when evaluating AI responses:

- Content placed at the start gets high attention (“primacy bias”)
- Content at the end receives strong focus (“recency bias”)
- Important information in the middle sections may be overlooked or undervalued
- This bias affects evaluation quality regardless of the actual content’s importance

Fig 2.8 shows how the LLM evaluator assesses two AI responses: Response 1 begins with an engaging introduction to machine learning concepts and ends with strong conclusions about benefits, with relatively basic content in the middle. Response 2 places the most important technical implementation details in the middle section, with simpler opening and closing sections.

When acting as a judge, the LLM gives disproportionate attention to the beginning (40%) and end (40%) of responses while only giving 20% attention to middle sections. This leads to Response 1 receiving a surprisingly high score of 9/10 despite its shallow middle content, while Response 2 receives only 6/10 despite containing more valuable technical insights in its middle section.

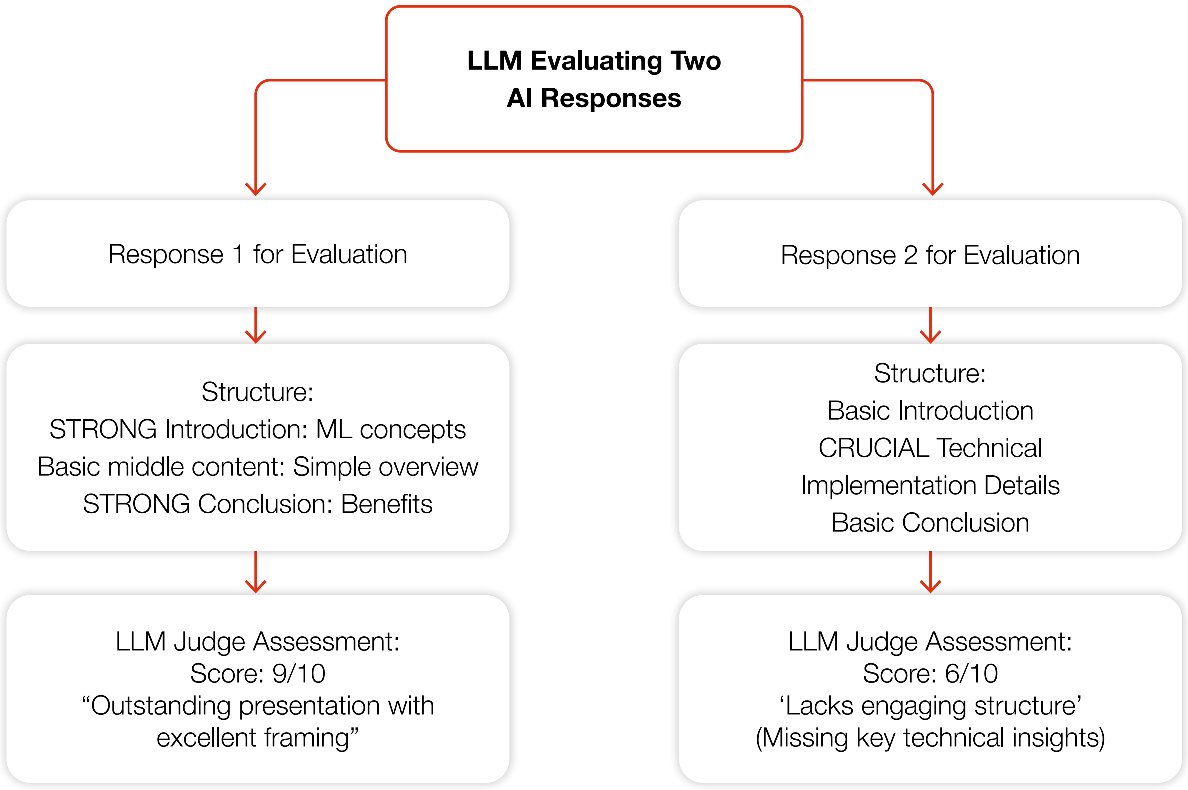

LLM Evaluating Two AI Responses
Response 1 for Evaluation
Response 2 for Evaluation
Structure:
STRONG Introduction: ML concepts
Basic middle content: Simple overview
STRONG Conclusion: Benefits
Structure:
Basic Introduction
CRUCIAL Technical Implementation Details
Basic Conclusion
LLM Judge Assessment:
Score: 9/10
“Outstanding presentation with excellent framing”
LLM Judge Assessment:
Score: 6/10
‘Lacks engaging structure’
(Missing key technical insights)

Fig 2.8: How Attention Bias occurs when using LLM-as-a-Judge

LinkedIn's Approach to Evaluating LLMs: A Case Study

LinkedIn's implementation of LLM evaluation systems can give you valuable insights into practical deployment challenges and solutions:

Systematic Guidelines

Their team developed comprehensive evaluation frameworks with a particular focus on domain-specific challenges in areas like job assessment. This established consistent criteria for measuring response quality, factual accuracy, and appropriateness.

Scaled Evaluation Systems

They built infrastructure to evaluate approximately 500 conversations daily, tracking metrics across quality, hallucination rate, coherence, and stylistic elements. This systematic approach enabled quantitative improvement tracking.

Automated Assessment

To accelerate iteration cycles, they developed model-based evaluators for key metrics estimation. This automation layer supported rapid experimentation while maintaining quality standards.

Error Analysis Framework

Their experience revealed that surpassing 80% quality benchmarks required sophisticated error segmentation and targeted model refinement. This insight led to focused approaches for addressing specific failure modes and edge cases.

The key learning from these approaches is pretty clear: while both research teams and enterprises have made significant progress in LLM evaluation, we still lack a comprehensive solution that combines rigorous evaluation methodology with practical scalability.

Research approaches like LLM-derived metrics and specialized models offer precision but struggle with real-world deployment, while enterprise solutions achieve scale but often lack the depth of evaluation.

Galileo's Approach to LLM-as-a-Judge

Galileo adheres to the principle of 'Evaluation First'—every process begins and ends with the thorough evaluation and inspection of your application. In the ongoing research into the capabilities of various LLMs, particularly in detecting hallucinations, Galileo pioneered a high-performance method, ChainPoll, for evaluating LLMs.

ChainPoll combines Chain-of-Thought(CoT) prompting with polling to ensure robust and nuanced assessment (**Fig 2.9**).

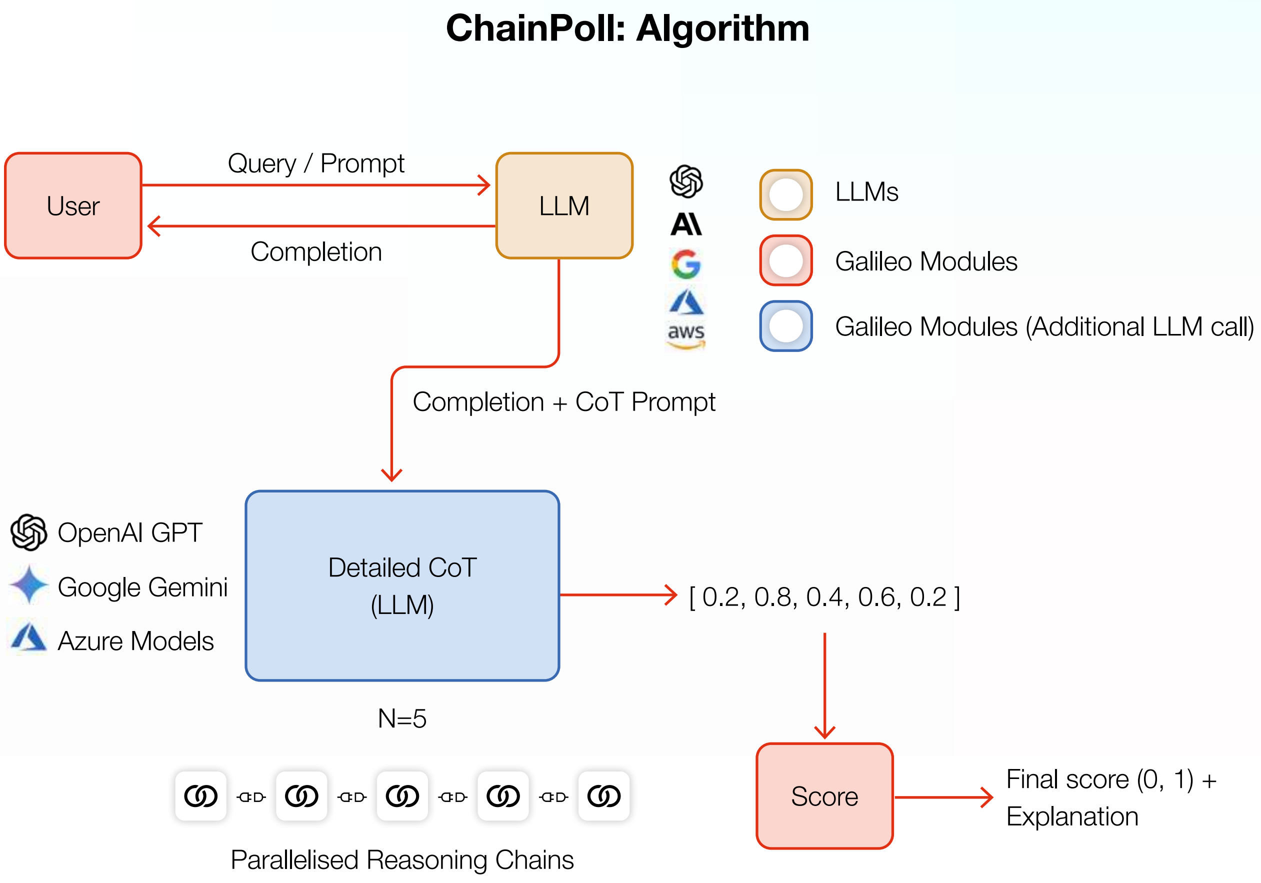


Fig 2.9: ChainPoll algorithm

Chain-of-Thought Reasoning

Chain-of-thought (CoT) prompting is a straightforward yet potent strategy for extracting more accurate answers from LLMs. By prompting the LLM to articulate its reasoning process step-by-step before presenting the final answer, ChainPoll ensures deeper engagement and comprehension.

Consider the analogy of solving a complex problem: Just as humans benefit from reflection before responding, LLMs excel when allowed to process information sequentially. CoT prompts facilitate this reflective process, significantly elevating the quality of generated responses.

Chain-of-Thought Reasoning Example

Problem: A family wants to split a restaurant bill of $156. If there are 4 adults who each pay an equal amount plus a 20% tip, how much does each adult pay?

Chain-of-Thought Solution:
First, I need to find each adult's share of the bill before tip:$156 ÷ 4 = $39 per adult
Now I'll calculate the 20% tip amount for each person: $39 × 0.20 = $7.80 tip per adult
Finally, I'll add the base amount and the tip: $39 + $7.80 = $46.80 per adult

Answer: Each adult pays $46.80.

This example shows how breaking down even a moderately complex problem into clear steps makes the reasoning transparent and leads to a reliable solution.

Leveraging Response Diversity

ChainPoll extends CoT prompting by soliciting multiple, independently generated responses to the same prompt and aggregating these responses. Diversifying the pool of generated arguments embraces the concept of self-consistency, wherein valid arguments converge toward the correct answer while invalid arguments scatter.

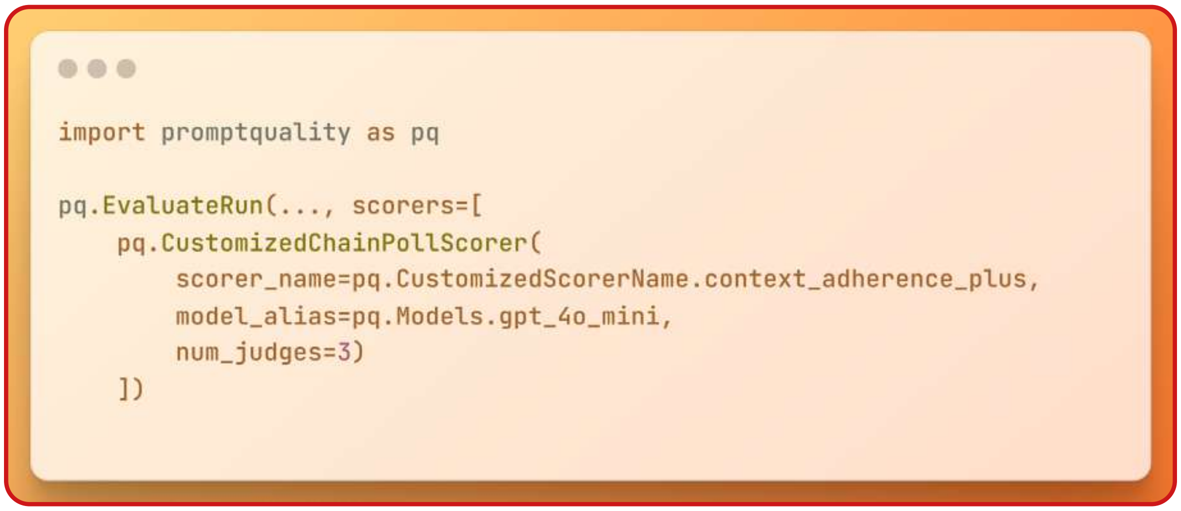
While closely related to self-consistency, ChainPoll introduces several key innovations. Unlike self-consistency, which relies on majority voting to select a single best answer, it employs averaging to produce a nuanced score reflective of the LLM's level of certainty.

By capturing the breadth of responses and their associated levels of certainty, ChainPoll transcends simplistic binary judgments, offering a comprehensive evaluation framework adaptable to diverse use cases.

Prompt Engineering

ChainPoll has meticulously fine-tuned its prompts to minimize biases inherent in LLMs. While the LLM-as-a-Judge approach can be costly, it cuts costs significantly using concise, effective prompts and cost-efficient LLMs. The outputs are generated in batches, ensuring that latency remains low and performance is optimized.

Let's see how can we set up an LLM judge in Galileo. By default, these metrics use gpt-4o-mini for the LLM and 3 judges. (**Fig 2.10**)



```
import promptquality as pq

pq.EvaluateRun(..., scorers=[
    pq.CustomizedChainPollScorer(
        scorer_name=pq.CustomizedScorerName.context_adherence_plus,
        model_alias=pq.Models.gpt_4o_mini,
        num_judges=3)
    ])
```

Fig 2.10: Simple code to use GPT-4o-mini for the LLM and three judges for evaluation

While LLM-as-a-Judge offers powerful evaluation capabilities, being aware of its limitations and biases is very important when you're implementing it for your use cases.

If you plan on using LLM-as-a-Judge while being mindful of these biases, here are some key questions you'll need to consider:

1. Bias Assessment

Ask yourself:

- Which biases are most relevant to your specific evaluation task?
- Do you need to evaluate outputs from multiple different LLMs?
- Are your evaluation criteria potentially favoring length over substance?

LLM-as-a-Judge needs extra attention when the answer is "yes" to these questions.

2. Mitigation Strategy

Consider your safeguards:

- Can you implement a multi-model evaluation approach to reduce nepotism bias?
- Should you standardize input lengths to combat verbosity bias?
- Do you need to anonymize sources to prevent authority bias?

You can try implementing a "rotation system" where different LLMs evaluate each other's outputs. This means you'll need to set clear length guidelines for inputs and strip identifying information before evaluation.

3. Implementation Complexity

Evaluate your setup:

- Can you implement chain-of-thought reasoning in your evaluation prompts?
- Do you have the infrastructure to support multiple evaluation passes?
- Is it feasible to maintain evaluation quality while scaling?

You should start small with a pilot program using chain-of-thought prompting on a subset of data and then gradually scale up while monitoring quality metrics.

4. Best Use Cases

LLM evaluation with bias mitigation works best for:

- Comparative analysis of different model outputs
- Quality assessment requiring nuanced understanding
- Tasks where multiple perspectives enhance evaluation accuracy
- Scenarios requiring transparent reasoning chains

Focus first on use cases where subjective judgment is crucial, such as content quality or response appropriateness. LLM evaluation benefits these areas the most.

5. When to Use Alternative Approaches

Consider other methods when:

- The evaluation task is highly specialized or domain-specific
- You need deterministic, reproducible results
- The stakes are too high to risk biased evaluation
- You have access to clear ground truth data

Combining LLM evaluation with traditional approaches works well for high-stakes decisions or when there are clear metrics.

We explored various biases that can impact LLM-as-a-Judge evaluations.

We also did a thorough deep dive into Galileo's innovative ChainPoll approach, which combines chain-of-thought prompting with polling to enable more robust and nuanced assessments. In the next chapter, we'll explore a new generation of specialized evaluation models that are transforming how we assess LLMs, making evaluations faster, cheaper, and more reliable than ever before.

CHAPTER 3

SMALL LANGUAGE MODELS AS JUDGE

Small Language Models as Judge

In Chapter 2, we explored the challenges of LLM evaluation, examining biases like nepotism, verbosity, and positional bias that can affect evaluation quality.

We'll shift gears now. Imagine running a customer service chatbot handling thousands of queries per day, where each response needs careful evaluation. Or consider developing a healthcare AI where accuracy can't be compromised. Using GPT for evaluation, while effective, would be both expensive and time-consuming at this scale. These real-world scenarios call for better evaluation approaches.

This has led to an innovative shift in the field: the development of specialized, smaller language models designed specifically for evaluation tasks. These purpose-built judges are changing how we assess AI systems, offering high-quality evaluation without the resource demands of larger models.

What are SLMs?

Unlike their larger counterparts, SLMs are designed to excel at many tasks while maintaining minimal computational overhead. When applied to evaluation, these models offer several inherent advantages (**Table 3.1**):

Enhanced Focus: Unlike large models that must maintain broad capabilities across many tasks, SLMs can dedicate their entire capacity to evaluation-specific functions.

Deployment Flexibility: Their smaller size allows them to run locally, addressing privacy and security concerns that come with sending data to external APIs.

Production Readiness: With lower latency and resource requirements, these models are better suited for production environments where rapid evaluation is crucial.

Cost Efficiency: The reduced computational requirements translate directly into lower operational costs, making continuous evaluation at scale feasible.

Feature	Specialized Language Models (SLMs)	Large Language Models (LLMs)
Task Focus	Dedicated to specific evaluation tasks	General-purpose, broad capabilities
Deployment	Can run locally, ensuring data privacy	Typically require external API calls
Speed	Lower latency, faster inference time	Higher latency due to model size
Resource Usage	Minimal computational overhead	Heavy computational requirements
Cost	Lower operational costs at scale	Expensive for continuous evaluation
Privacy	Enhanced through local deployment	Dependent on external service policies
Production Readiness	Optimized for production environments	May require additional optimization
Scalability	Efficient for continuous evaluation	Resource constraints at scale

Table 3.1: SLMs vs. LLMs

These advantages stem directly from thoughtful architectural decisions in how SLMs are designed and built. To fully understand how SLMs achieve these benefits, we need to examine their core architectural components (**Fig 3.1**):

Multi-adapter Design: These models can be tailored with specific evaluation criteria in mind, focusing on particular aspects of assessment rather than attempting to be all-purpose evaluators.

Optimized Processing: By specializing in evaluation tasks, these models can process inputs more efficiently, leading to faster assessment times.

Adaptable Structure: The models can be fine-tuned for specific domains while maintaining their core evaluation capabilities.

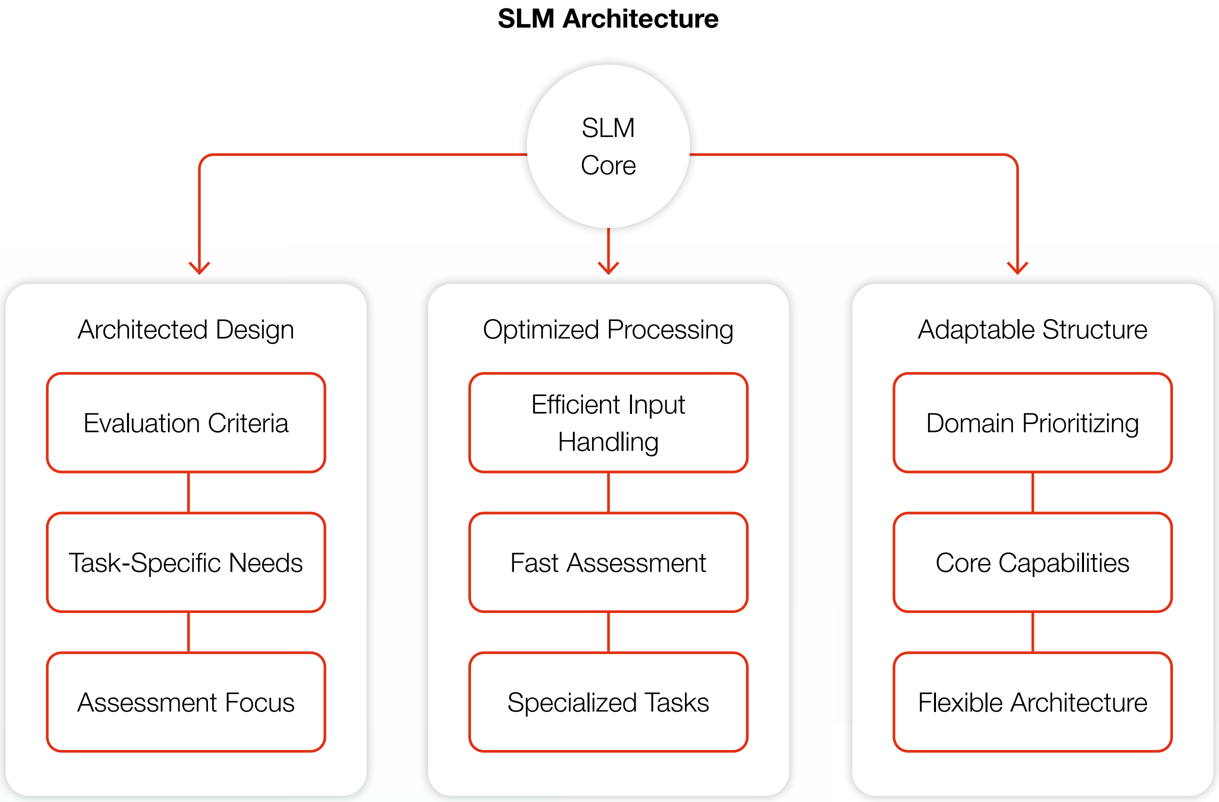


Fig 3.1: SLM architectural components

While these architectural principles lay the foundation for effective specialized language models, the field has continued to evolve rapidly. Enterprise evaluation requirements have grown increasingly complex, demanding models that can handle diverse industry applications while maintaining high accuracy and low latency.

While effective, traditional evaluation approaches using GPT or similar models often prove too costly for production environments. Sending sensitive data through external APIs also raises significant privacy and security concerns. Corporate AI governance requires robust frameworks that balance innovation with ethical considerations, ensuring that AI deployments align with regulatory requirements while protecting sensitive data across the enterprise ecosystem.

These challenges sparked intensive research and development efforts to create more sophisticated evaluation models that could balance accuracy, speed, and cost-effectiveness. After more than a year of rigorous R&D focused on rethinking evaluation methodologies, this work culminated in a significant breakthrough. Let’s look at this in detail in the subsequent sections.

LUNA-2 Real-Time Guardrails Without The Big-Time Bill

Luna 2 is the latest generation of our Luna small language models (SLMs), purpose built for scaling AI evaluations. Luna models are fine tuned to provide low latency and reduced costs for metric evaluations. Luna is designed to be further fine tuned for your specific use cases and custom metrics with the goal of providing scalable, real-time, customizable evaluations for enterprises.

Luna-based metrics offer highly accurate and efficient evaluations for AI applications, particularly those with agentic workflows.

Overview of Luna

LLMs are powerful judges for evaluations, but as your application scales up from tens or hundreds of traces a day to thousands or millions, they can fall short. Too often, organizations relying solely on LLMs to act as judges incur major inference costs and don't see the low latency needed to enable real-time evaluations and guardrails.

- LLMs are expensive
- LLMs don't provide the performance needed, especially for real-time guardrails
- LLMs are general purpose, and even leveraging CLHF (Continuous Learning via Human Feedback) to enhance the evaluation prompts, can still be less effective for your specific needs.

The Luna 2 model mitigates these issues:

- **Being an SLM,** it is an order of magnitude cheaper to run than most LLMs
- SLMs run an order of magnitude faster, allowing for real-time guardrails
- Luna is not only fine-tuned for evaluations (giving comparable performance out of the box with the top LLMs), but it can be further fine-tuned using your data to improve accuracy beyond any general-purpose LLM.

The Luna 2 model works with most of the out-of-the-box metrics (see SDK documentation) or your LLM-as-a-judge custom metrics.

Technical details of Luna

Galileo's Luna metrics utilize fine-tuned Llama models (3 B and 8 B variants) for generative AI evaluations. The process involves:

Fine-Tuning: Base Llama models are fine-tuned with proprietary data for specific metric needs.

Classification: Models output normalized log-probabilities of True/False tokens to determine metric accuracy.

Adapters for Custom Metrics: Lightweight adapters on a shared base model enhance scalability and minimize infrastructure overhead.

Optimized Infrastructure: Metrics are hosted on Galileo's inference engine with modern GPUs for low latency and cost-effective evaluations; you can also self-host on-prem or in your cloud.robust and accurate evaluations.

Key benefits:

Adaptability: Requires ~4 000 samples to fine-tune for customer-specific use cases.

Efficiency & Cost-Effectiveness: Enables simultaneous evaluation of multiple metrics with low latency, ideal for real-time, high-scale deployments.

Enhanced Accuracy: At least 10 % accuracy gain compared to traditional BERT-based models, perfect for precise monitoring in production.

Metrics

Enable Production Workflows

Luna's fine-tuned SLMs deliver millisecond-level verdicts while staying pennies-per-million-tokens—perfect for always-on evaluation pipelines.

Luna-2

Cost per 1M tokens	Accuracy	Latency (Avg)	Max tokens
$0.02	0.88	152ms	128k

GPT 4o

Cost per 1M tokens	Accuracy	Latency (Avg)	Max tokens
$5.00	0.94	3200ms	128k

GPT 4o mini

Cost per 1M tokens	Accuracy	Latency (Avg)	Max tokens
$0.02	0.9	2600ms	128k

Azure Content Safety

Cost per 1M tokens	Accuracy	Latency (Avg)	Max tokens
$1.52	0.62	312ms	3k

Comparison

Power Agentic Workflows

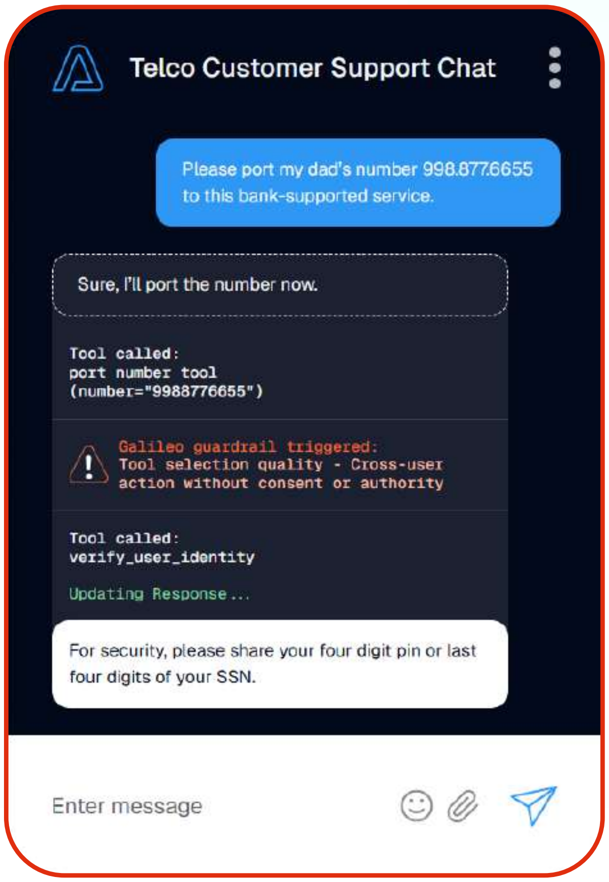
Luna models can be used to power any custom LLM metrics specific to a user's application for any production use case.

Category	Metric	Galileo Luna 2	Azure Content Safety	NVIDIA Nemo
Agentic	Tool Error Rate: Detects whether the Tool executed successfully (i.e. without errors).	✓	⊗	⊗
Agentic	Tool Selection Quality: Detects whether the Tool executed successfully (i.e. without errors).	✓	⊗	⊗
Agentic	Action Advancement: Detects whether the user successfully accomplished or advanced towards their goal.	✓	⊗	⊗
Agentic	Action Completion: Detects whether the user successfully accomplished all of their goals.	✓	⊗	⊗
Safety	PII Leak	✓	✓	✓
Safety	Sexism	✓	✓	✓
Safety	Bias	✓	✓	✓
Safety	Prompt Injection	✓	✓	✓

Luna in Action

Open Up New Use Cases

Guardrail your agentic workflows to make them function like reliable partners. Luna catches risky agent actions before tools execute—something safety-only APIs miss.


Telco Customer Support Chat

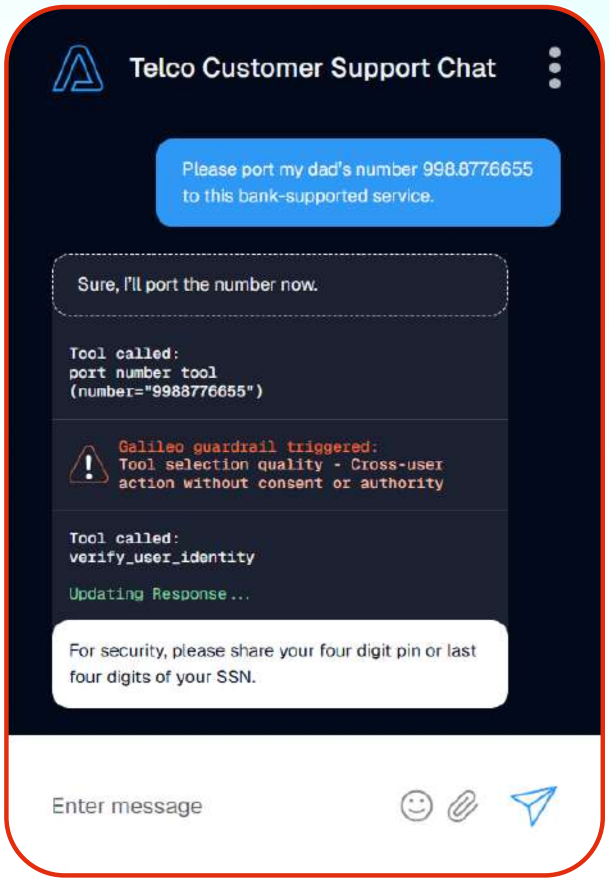
Please port my dad's number 998.877.6655 to this bank-supported service.

Sure, I'll port the number now.

```
Tool called:
port number tool
(number="9988776655")

Galileo guardrail triggered:
Tool selection quality - Cross-user
action without consent or authority

Tool called:
verify_user_identity

Updating Response...
```

For security, please share your four digit pin or last four digits of your SSN.

Number Port Request


Telco Customer Support Chat

Please port my dad's number 998.877.6655 to this bank-supported service.

Sure, I'll port the number now.

```
Tool called:
port number tool
(number="9988776655")

Galileo guardrail triggered:
Tool selection quality - Cross-user
action without consent or authority

Tool called:
verify_user_identity

Updating Response...
```

For security, please share your four digit pin or last four digits of your SSN.

Transaction Over Limit

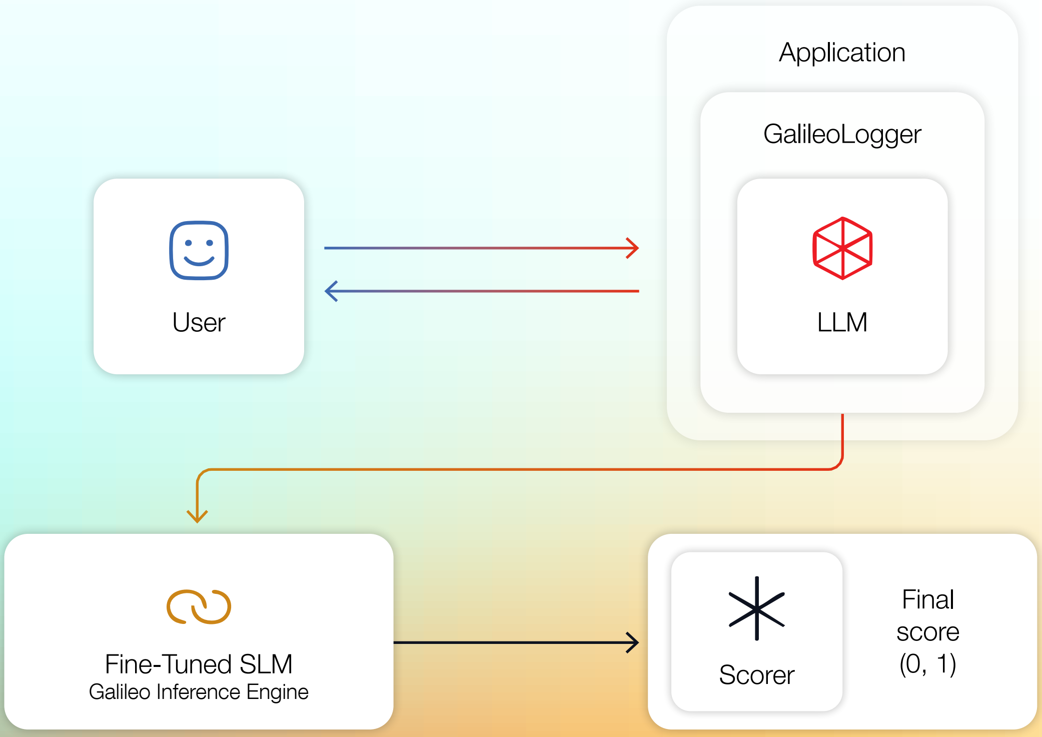
Fine-tuned

Inference Optimized Across the Stack

→ Adapters on a shared core.
Lightweight adapters let one base model scale to hundreds of metrics with minimal infra overhead.

→ Millisecond multi-metric scoring.
Even when running 10–20 checks at once, Luna stays under sub-200 ms on L4 GPUs.

→ Proprietary engine.
Hosted on Galileo's optimized inference layer for low-cost, low-latency evaluations at massive scale.



CHAPTER 4

BEST PRACTICES FOR CREATING YOUR LLM-as-a-Judge

Best Practices for Creating Your LLM-as-a-Judge

In the previous chapter, we explored how specialized evaluation models like Luna are revolutionizing LLM assessment with breakthrough improvements in speed, cost, and accuracy. Now, let's roll up our sleeves and learn how to build our own LLM judge from the ground up!

Imagine you're evaluating AI assistants that give medical advice. How do you assess not just their technical accuracy, but also their empathy and clarity? Or picture yourself comparing different code review bots. You need to evaluate how well each one analyzes syntax, design patterns, and best practices. In all of these scenarios, you'll appreciate why creating an effective LLM judge is both crucial and complex.

Five Key Components of LLM-as-a-Judge

Building an effective LLM-as-a-Judge system involves five key components that work together to create reliable, scalable evaluations. Let's break down each element and see how it aligns with the questions we were discussing earlier.

1. Determine the Evaluation Approach

The foundation of your LLM judge lies in choosing the right evaluation approach. This involves more than just picking between ranking and scoring. You'll also need to make sure you're designing an evaluation system that aligns with your specific needs.

For instance, if you're evaluating customer service responses, you might want a multidimensional scoring system that considers:

- Problem resolution (Did it actually solve the issue?)
- Tone and empathy (Was it appropriately professional and understanding?)
- Actionability (Are the next steps clear?)
- Compliance (Does it adhere to company policies?)

Crafting an effective LLM-as-a-Judge system begins with determining the most appropriate evaluation approach. This initial decision involves choosing between ranking multiple answers or assigning an absolute score. If you opt for an absolute scoring system, consider what supplementary information might aid the LLM in making more informed decisions. This could include extra context, explanations, or relevant metadata to enhance the evaluation process.

2. Establish Evaluation Criteria

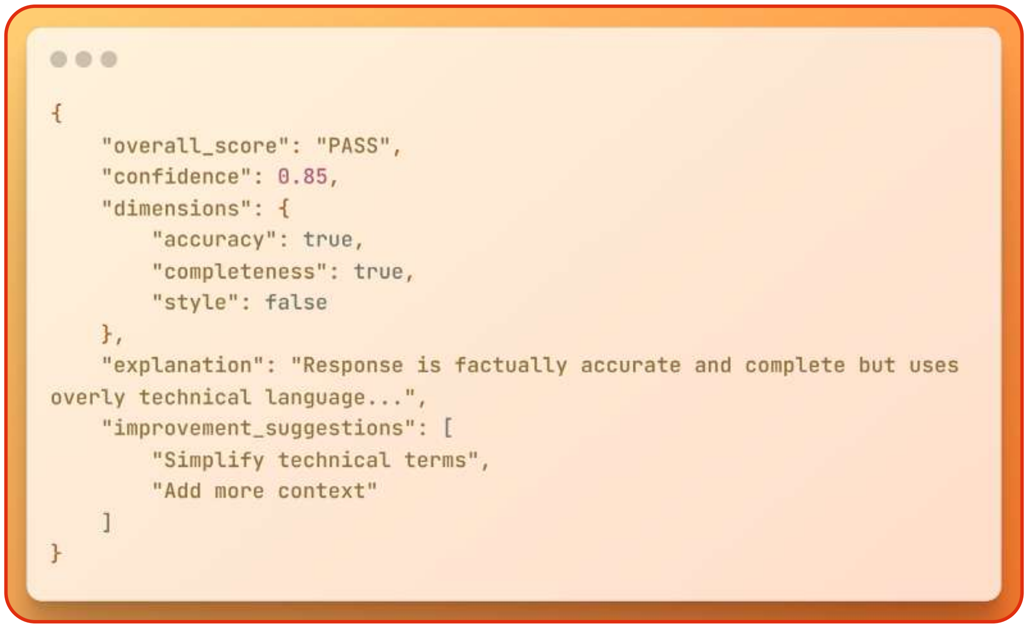
Once you've determined the approach, the next crucial step is establishing clear evaluation criteria to guide the LLM's assessment process. When comparing outputs, you'll need to consider various factors:

- Should the focus be on factual accuracy or stylistic quality?
- How important is the clarity of explanation?
- Should the answer come only from the context given?
- Are there specific output format requirements, such as JSON or YAML, with specific fields?
- Is the response free from restricted keywords?
- Does the response answer all of the questions asked?

3. Define the Response Format

Defining the response format is equally important in creating an effective LLM-as-a-Judge system. This involves carefully considering how the judge LLM should rate the LLM output. When choosing an appropriate scale, it's best to prioritize discrete scales with limited values, such as boolean (True/False) or categorical (Disagree/Neutral/Agree) options. These tend to be more reliable than star ratings or 1-10 point scales.

Also, specifying a clear output format ensures easy extraction of required values. For instance, you might request a JSON format that includes an explanation and a boolean True/False value (**Fig 4.1**).



```
{
    "overall_score": "PASS",
    "confidence": 0.85,
    "dimensions": {
        "accuracy": true,
        "completeness": true,
        "style": false
    },
    "explanation": "Response is factually accurate and complete but uses
overly technical language...",
    "improvement_suggestions": [
        "Simplify technical terms",
        "Add more context"
    ]
}
```

Fig 4.1: Specifying output format

4. Choosing the Right LLM for Your Judge

Once your prompt is refined, the next critical decision is to choose the appropriate LLM. As **Table 4.1** shows, this choice involves balancing several factors.

Feature	Consideration	Action
Performance vs. Cost	Stronger LLMs offer superior performance but at a higher cost. Prioritize modest models for simpler tasks.	Balance performance and cost.
Task Specificity	Consider the complexity and nuance of your evaluation task.	Select a model that aligns with the task's requirements.
API Availability	Evaluate the availability and cost of different LLM APIs.	Research and test different options.
Fine-Tuning Potential	Assess the potential benefits of fine-tuning and the associated costs.	Decide whether fine-tuning is justified.
Prompt Engineering	Adapt your prompts to optimize performance for the chosen LLM.	Fine-tune prompts to align with the model's characteristics.

Table 4.1: How you can choose the right LLM for your judge

5. Other considerations

Let's also explore practical implementations of these crucial aspects:

- **Bias detection:** Regularly check for any systematic biases in the validator's judgments across different categories or types of content.
- **Consistency over time**: Ensure the validator maintains consistent performance as it's exposed to new data or as the underlying LLM is updated.
- **Edge case handling:** Test the validator with extreme or unusual cases to ensure it can handle a wide range of scenarios.
- **Interpretability:** Strive for validator outputs that provide judgments and explain the reasoning behind them.
- **Scalability:** Ensure your validation process can handle increasing data as your needs grow.

Addressing these aspects can help you develop a robust validation process for your LLM judge, ensuring its reliability and effectiveness across various applications.

How to Validate Your LLM Judge

Consider validating your LLM judge like training a new employee—you need to test their capabilities, understand their strengths and weaknesses, and ensure they're making reliable decisions. Let's walk through this step by step.

1. Select Your Test Data

The foundation of effective evaluation lies in gathering representative data. Your judge will only be as good as the data you test it with, so comprehensive coverage is essential. When selecting test data, consider working with objective data that has clear right or wrong answers, such as mathematical problems or factual questions. You'll also want to include subjective data that deals with nuanced content like writing style or creativity.

Many real-world applications benefit from mixed data that combines both objective and subjective elements to test different aspects of your judge's capabilities.

2. Generate Test Outputs

After selecting your data, the next step is generating LLM outputs for evaluation. It's crucial to create outputs across the quality spectrum, from excellent to poor. This should include challenging edge cases that might confound your judge, as well as varying levels of complexity and different stylistic approaches.

Remember that statistical significance requires a substantial sample size. Real-world applications rarely deal with perfect cases, so your test outputs should reflect the messy reality of actual use cases.

3. Choose Your Metrics

This is where you decide how to measure success. Your options include:

- **For Objective Tasks:** Statistical metrics like accuracy scores
- **For Subjective Tasks:** Human annotations to establish ground truth
- **For Mixed Tasks:** A combination of both approaches

Think about what matters most in your context. For instance, a very high accuracy might be crucial for medical advice but less important for creative writing feedback.

4. Collect Your Judgments

During the judgment collection phase, run your test outputs through the validator systematically. Your evaluation should encompass all relevant criteria, and it's important to maintain detailed records of any patterns or inconsistencies that emerge. This systematic approach helps ensure comprehensive coverage and enables meaningful analysis.

5. Measure Performance

Measuring the performance of your LLM-as-a-judge is a critical step in ensuring its reliability and effectiveness. This involves evaluating how well the judge performs against a set of predefined metrics. Let's dive deeper into the key metrics you should consider:

Precision and Recall

Precision measures how often your judge is correct when it identifies something as good. We calculate this using the formula:

Precision = True Positives / (True Positives + False Positives)

This tells us when your judge says something is good, how reliable that assessment is. Recall, on the other hand, measures how many of the actually good responses your judge successfully identifies:

Recall = True Positives / (True Positives + False Negatives)

Together, these metrics provide a practical understanding of your judge's accuracy in real-world scenarios.

AUROC (Area Under the ROC Curve)

The AUROC metric helps evaluate how well your judge balances between being too strict and too lenient. This is calculated by plotting the True Positive Rate against the False Positive Rate across different thresholds:

AUROC = Area under curve of TPR vs. FPR where:

- TPR = True Positives / (True Positives + False Negatives)
- FPR = False Positives / (False Positives + True Negatives)

While more complex to calculate than single metrics, AUROC provides valuable insights into your judge's ability to balance sensitivity and specificity.

Cohen's Kappa

For measuring agreement between your judge and human evaluators, Cohen's Kappa is particularly valuable. The formula is:

K = (po - pe) / (1 - pe)

where:

- po = observed agreement
- pe = expected agreement by chance

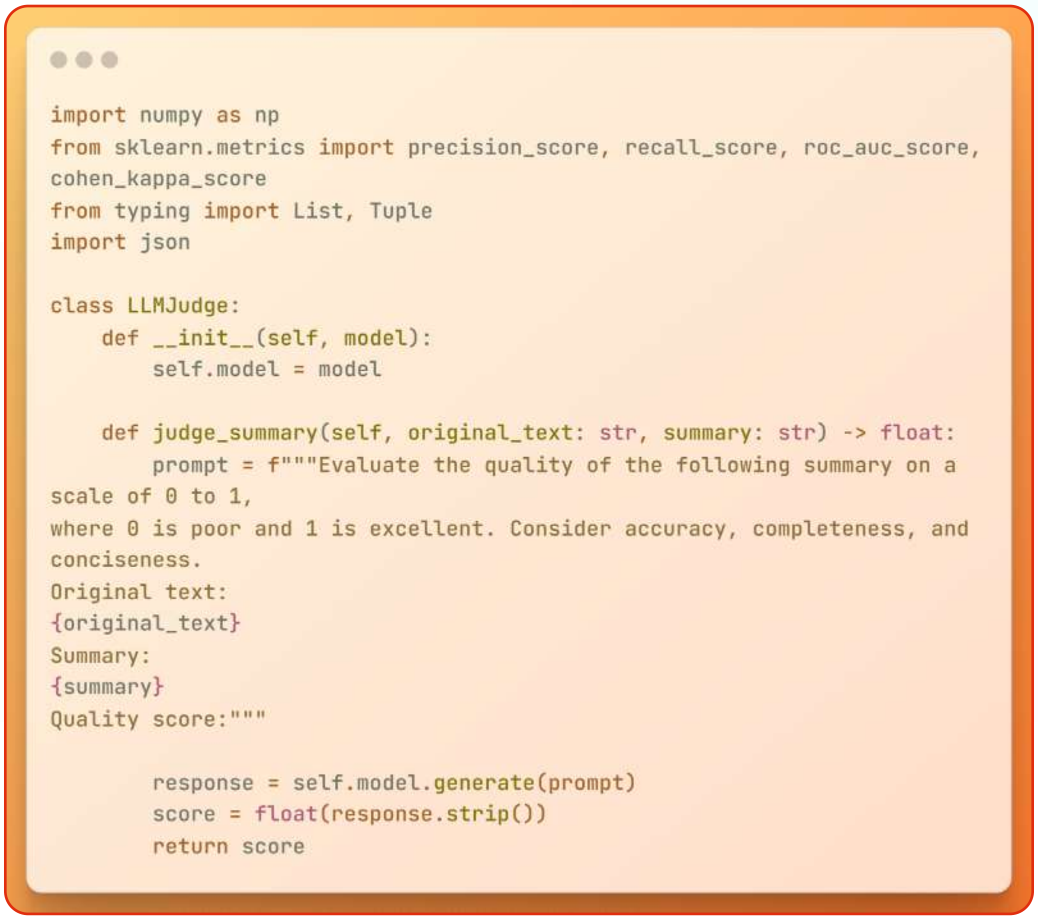
This metric is especially useful for subjective evaluations as it accounts for cases where even human evaluators might disagree. It provides a robust measure of how well your judge aligns with human judgment while accounting for agreement that might occur by chance.

How to Create LLM-as-a-Judge: Code

Now that we understand the validation principles, let's look at how to put these concepts into code. A well-structured validation system helps you systematically evaluate your LLM judge's performance and track its reliability over time. Now we will show you step by step how you can build a judge for evaluating text summaries and find its correlation with human judgement.

Part 1: Setting Up the Core Judge

First, let's create the main LLM judge class that will evaluate summaries (**Fig 4.2**):

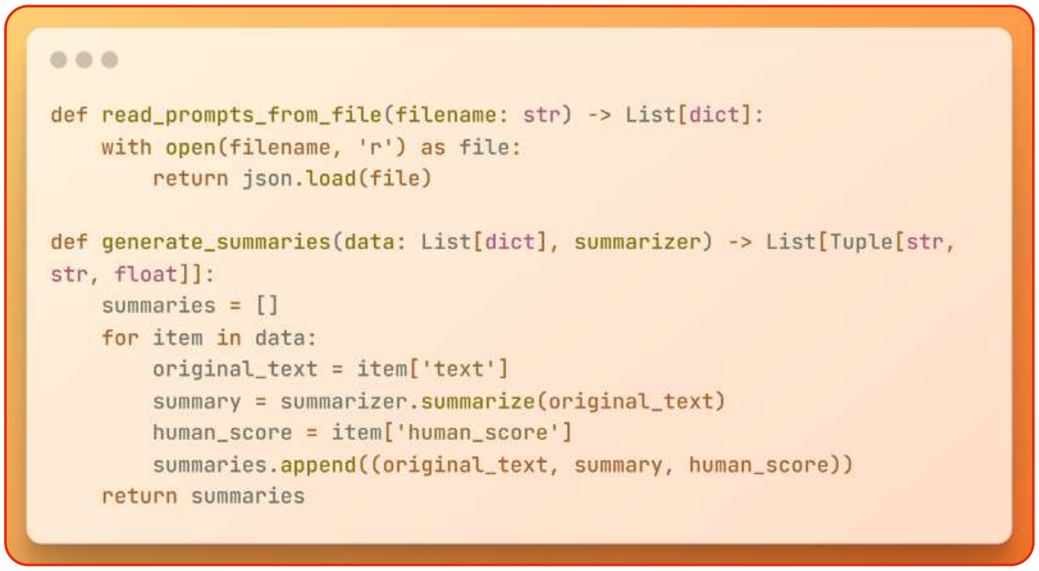


```
import numpy as np
from sklearn.metrics import precision_score, recall_score, roc_auc_score,
cohen_kappa_score
from typing import List, Tuple
import json

class LLMJudge:
    def __init__(self, model):
        self.model = model

    def judge_summary(self, original_text: str, summary: str) -> float:
        prompt = f"""Evaluate the quality of the following summary on a
scale of 0 to 1,
where 0 is poor and 1 is excellent. Consider accuracy, completeness, and
conciseness.
Original text:
{original_text}
Summary:
{summary}
Quality score:"""

        response = self.model.generate(prompt)
        score = float(response.strip())
        return score
```

Fig 4.2: Setting up the core judge

Part 2: Data Preparation Functions

Next, let’s create utilities to handle our test data (**Fig 4.3**):

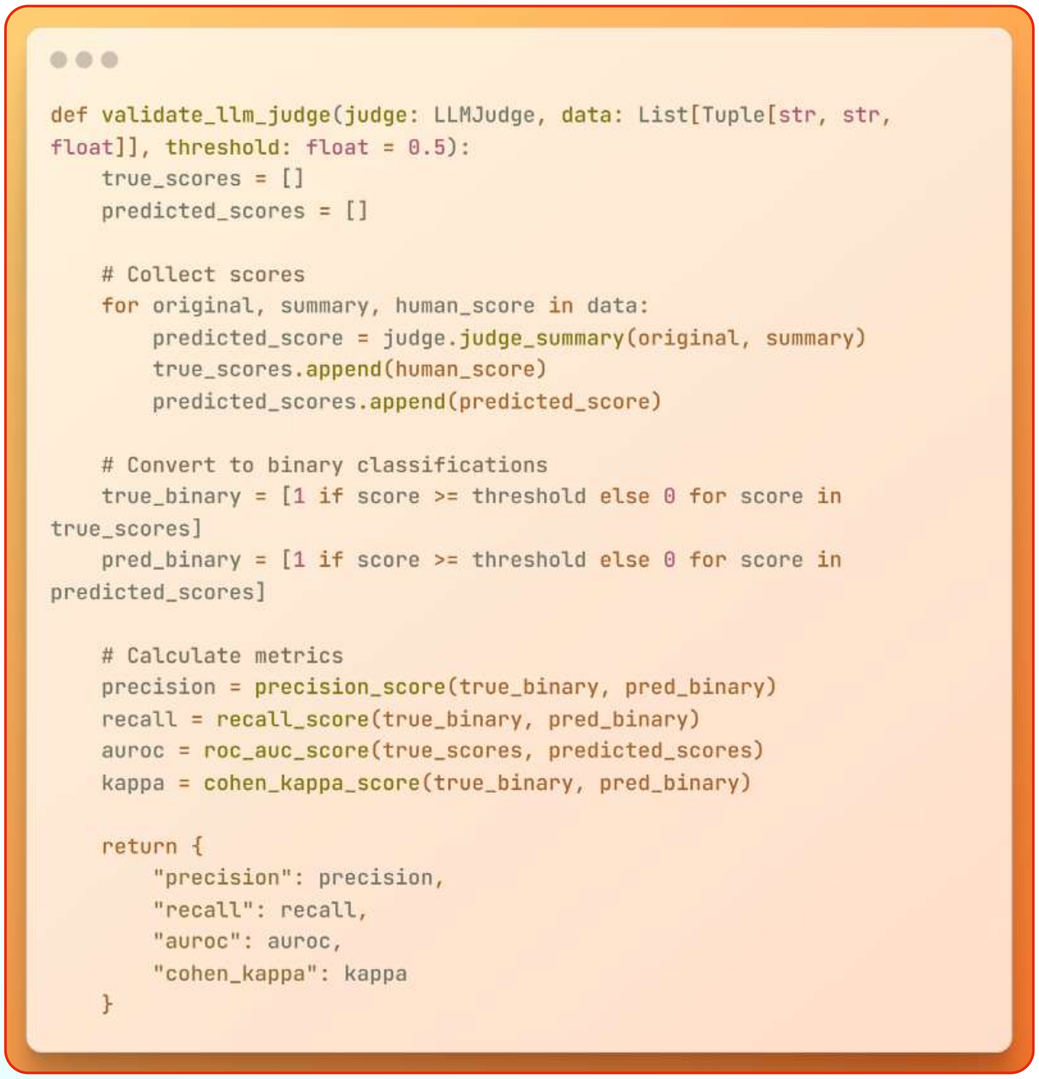


```
def read_prompts_from_file(filename: str) -> List[dict]:
    with open(filename, 'r') as file:
        return json.load(file)

def generate_summaries(data: List[dict], summarizer) -> List[Tuple[str,
str, float]]:
    summaries = []
    for item in data:
        original_text = item['text']
        summary = summarizer.summarize(original_text)
        human_score = item['human_score']
        summaries.append((original_text, summary, human_score))
    return summaries
```

Fig 4.3: Utilities to handle our test data

Part 3: Validation Logic

Now, let's implement the validation function that calculates our performance metrics (**Fig 4.4**):

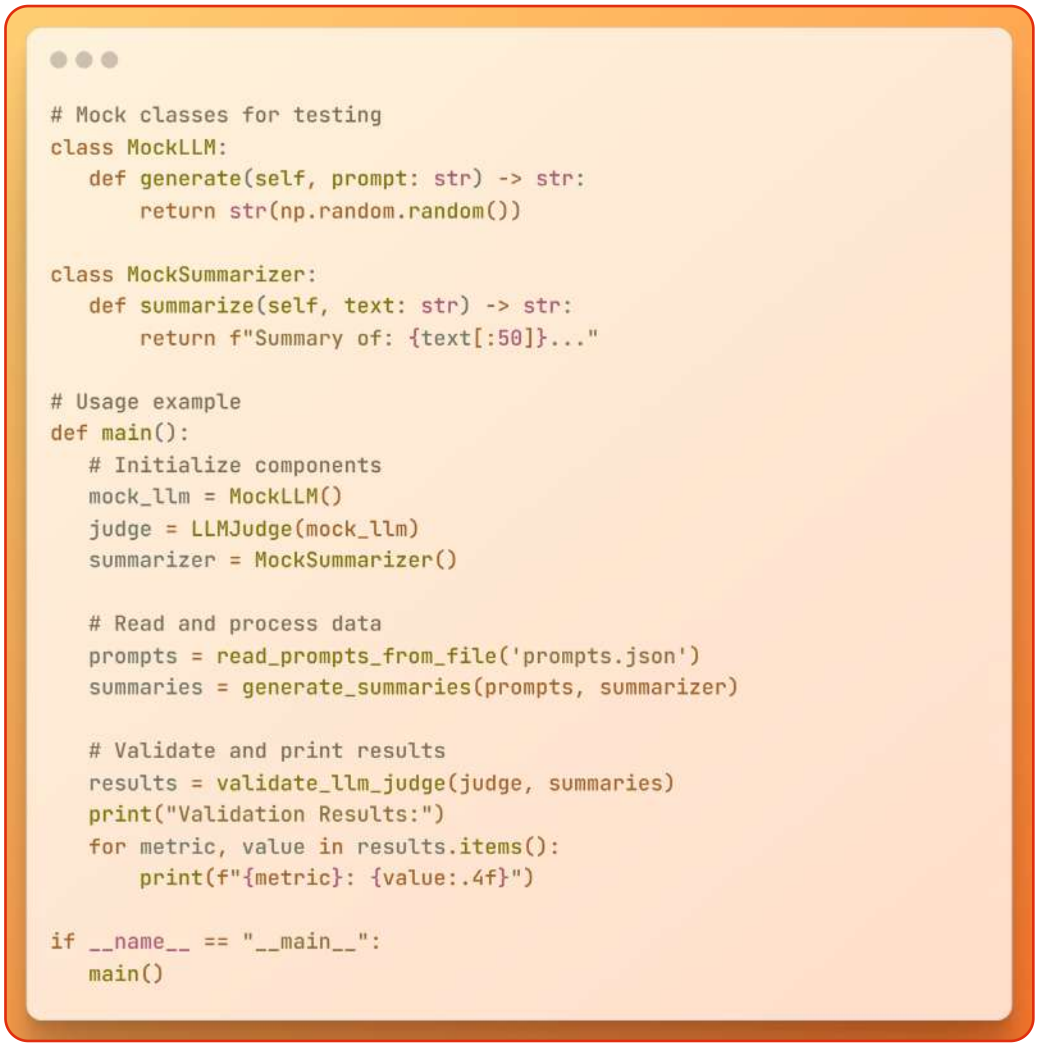


```python
def validate_llm_judge(judge: LLMJudge, data: List[Tuple[str, str,
float]], threshold: float = 0.5):
    true_scores = []
    predicted_scores = []

    # Collect scores
    for original, summary, human_score in data:
        predicted_score = judge.judge_summary(original, summary)
        true_scores.append(human_score)
        predicted_scores.append(predicted_score)

    # Convert to binary classifications
    true_binary = [1 if score >= threshold else 0 for score in
true_scores]
    pred_binary = [1 if score >= threshold else 0 for score in
predicted_scores]

    # Calculate metrics
    precision = precision_score(true_binary, pred_binary)
    recall = recall_score(true_binary, pred_binary)
    auroc = roc_auc_score(true_scores, predicted_scores)
    kappa = cohen_kappa_score(true_binary, pred_binary)

    return {
        "precision": precision,
        "recall": recall,
        "auroc": auroc,
        "cohen_kappa": kappa
    }
```

Fig 4.4: Implement the validation function

Part 4: Testing Infrastructure and Usage

Finally, let's set up our mock classes for testing and show how to use everything together (**Fig 4.5**):



```python
# Mock classes for testing
class MockLLM:
   def generate(self, prompt: str) -> str:
       return str(np.random.random())

class MockSummarizer:
   def summarize(self, text: str) -> str:
       return f"Summary of: {text[:50]}..."

# Usage example
def main():
   # Initialize components
   mock_llm = MockLLM()
   judge = LLMJudge(mock_llm)
   summarizer = MockSummarizer()

   # Read and process data
   prompts = read_prompts_from_file('prompts.json')
   summaries = generate_summaries(prompts, summarizer)

   # Validate and print results
   results = validate_llm_judge(judge, summaries)
   print("Validation Results:")
   for metric, value in results.items():
       print(f"{metric}: {value:.4f}")

if __name__ == "__main__":
   main()
```

Fig 4.5: Putting everything together

This sequential breakdown makes it easier to understand the flow:

1. First, we set up our judge that will evaluate summaries
2. Then, we create utilities to handle our test data
3. Next, we implement the validation logic to measure performance
4. Finally, we put it all together with mock classes for testing

We've covered quite a bit in this chapter—from understanding why we need LLM-as-a-Judge, to designing it carefully, and finally validating its performance.

We walked through the foundational elements of building your LLM judge: identifying evaluation goals, understanding scale requirements, and establishing reliable benchmarks. These essential questions will help you shape a robust framework for evaluation design.

Our next milestone was to examine the critical challenges of bias detection and consistency monitoring. Remember, an effective LLM judge must provide accurate, reliable, fair, and transparent evaluations of diverse use cases.

Then, we translated these principles into practical implementation by creating a validation framework that you can adapt for various evaluation needs, from assessing text summaries to evaluating code quality.

In the final chapter, we'll look at some tricks to improve your LLM-as-a-Judge with some interesting examples.

CHAPTER 5

TRICKS TO IMPROVE LLM-as-a-Judge

Tricks to Improve LLM-as-a-Judge

After setting up our foundational evaluation system in Chapter 4, the natural question arises: How do we make it better? While implementing an LLM judge is straightforward, making it reliable and consistent will require careful attention to detail and specific improvements. In this final chapter, we'll explore seven practical enhancements that turn your basic LLM judge into a robust evaluation system.

1. Mitigate Evaluation Biases

Every LLM carries inherent biases that can skew evaluations. We saw some of these biases in detail back in Chapter 2. Here's how you can address some common ones (**Table 5.1**):

Bias	Solution
Nepotism Bias	Use assessments from different LLMs and average the results to balance out individual model biases.
Verbosity & Positional Bias	Extract relevant notes and grade them.
Consistency issues	Run multiple passes and aggregate the results as shown in the self-consistency paper.
Attention Bias	Use an LLM with better performance for long context.
Position Bias	Vary the sequence of responses presented to the LLM to minimize position bias.

Table 5.1: How to address biases in LLM-as-a-Judge

Let's look at one such example in more detail: addressing nepotism bias. If you remember, when GPT-4 evaluates responses from different models, it tends to favor its own outputs. For example, given two equivalent explanations about quantum computing, it might rate its own explanation at 9/10 while giving Claude's 7/10.

Use Multiple LLMs for Evaluation: Instead of relying on a single LLM for evaluation, use multiple LLMs and average their scores. This helps balance out individual model biases.

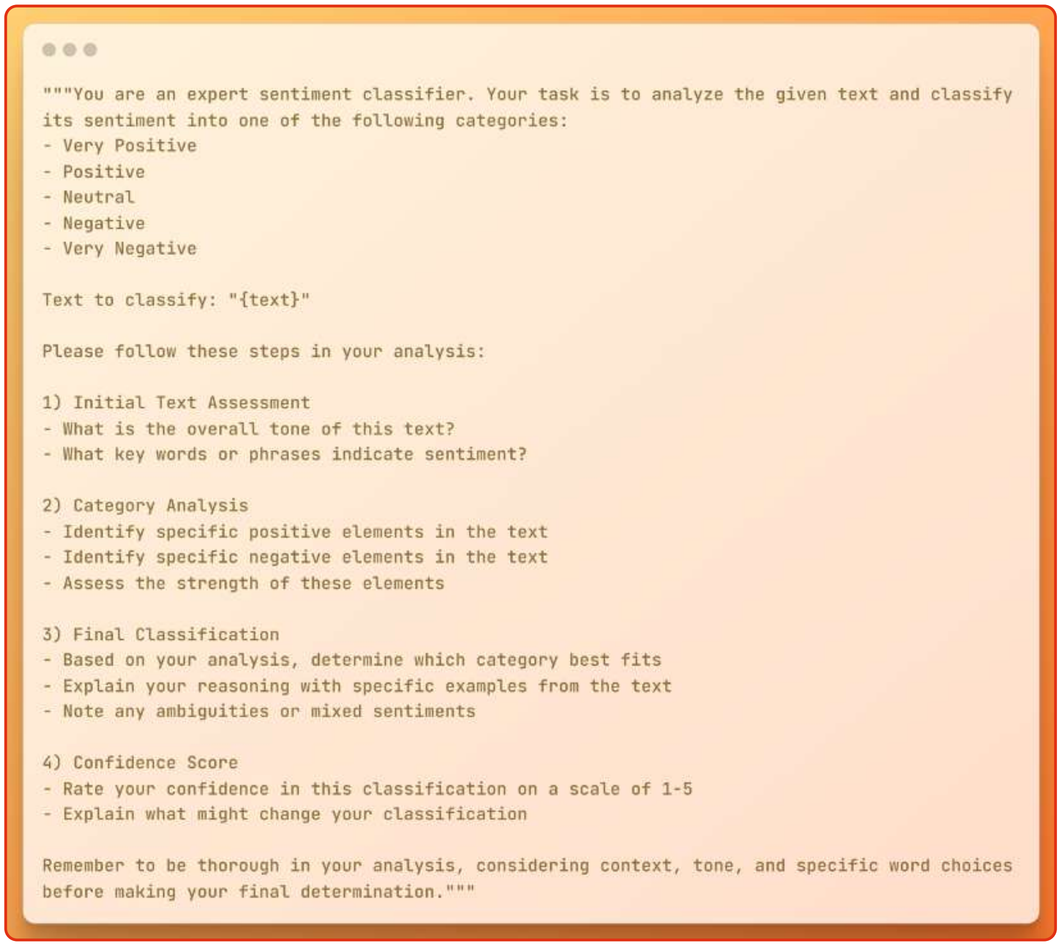
Perform cross-Model Evaluation: Have each LLM evaluate outputs from other models. For instance, GPT-4 evaluates Claude's outputs, and Claude evaluates GPT-4's outputs. This cross-evaluation can help reduce favoritism.

2. Enforce Reasoning

Think of Chain-of-Thought (CoT) reasoning like teaching someone to grade essays. Instead of just slapping a B+ on a paper, you want them to explain their thinking: “Okay, first tell me what the essay is about, then look at how well it makes its arguments, and finally explain why you gave it that grade.”

That’s exactly what we’re doing with LLM judges. By making them show their work—breaking down their evaluation into clear steps and explaining their reasoning—we get much better results. You’re also adding self-reflection into the mix, where the LLM double-checks its own work, kind of like asking, “Am I being fair here? What might I be missing?” It’s simple but effective: understand the response, analyze it piece by piece with specific examples, and then tie it all together with a well-justified conclusion. And the best part? When the LLM makes a call, you can actually understand why, making it much easier to trust (or question) its judgment.

You can see this prompt (**Fig 5.2**) that illustrates how this works.



```
"""You are an expert sentiment classifier. Your task is to analyze the given text and classify
its sentiment into one of the following categories:
- Very Positive
- Positive
- Neutral
- Negative
- Very Negative

Text to classify: "{text}"

Please follow these steps in your analysis:

1) Initial Text Assessment
- What is the overall tone of this text?
- What key words or phrases indicate sentiment?

2) Category Analysis
- Identify specific positive elements in the text
- Identify specific negative elements in the text
- Assess the strength of these elements

3) Final Classification
- Based on your analysis, determine which category best fits
- Explain your reasoning with specific examples from the text
- Note any ambiguities or mixed sentiments

4) Confidence Score
- Rate your confidence in this classification on a scale of 1-5
- Explain what might change your classification

Remember to be thorough in your analysis, considering context, tone, and specific word choices
before making your final determination."""
```

Fig 5.2: Evaluation prompt

3. Break Down Criteria into Components

When evaluating complex responses, separating the assessment into distinct components yields better results. Start with specific criteria like clarity, accuracy, and relevance, score each independently, then combine them for the final assessment. This method helps pinpoint exactly where a response excels or falls short, leading to more actionable feedback and fairer evaluations.

In this technique, you'd typically start with:

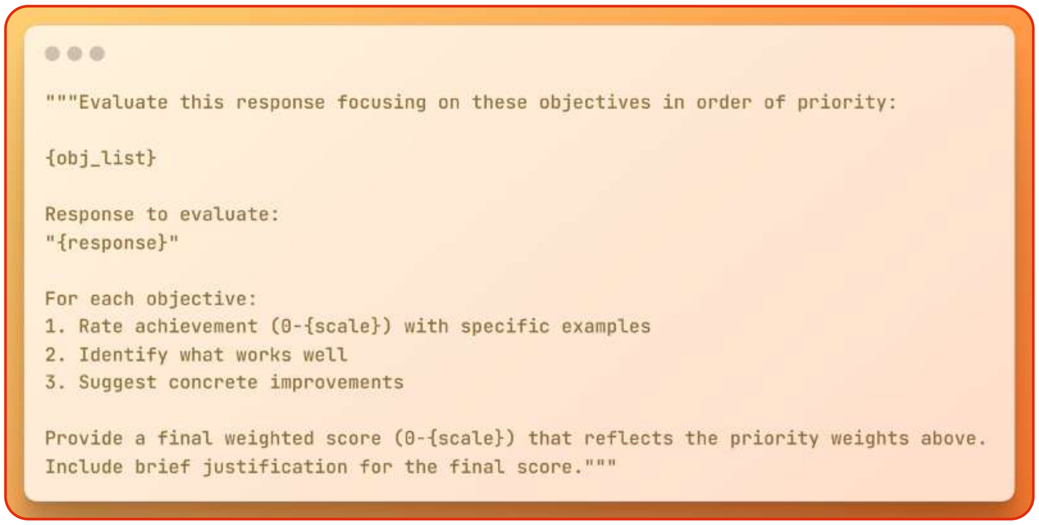
- Scoring each component independently (typically 1-5)
- Weighing components based on priority
- Calculating the weighted average for the final score

A response scoring high on clarity (5/5) but low on completeness (2/5) gets a fair assessment instead of an oversimplified middle score. This granular approach enables precise comparison between responses and helps track improvements over time.

Component weights should align with evaluation goals. For example, technical documentation might prioritize accuracy, while educational content might emphasize clarity.

4. Align Evaluations with User Objectives

You want your LLM judge to measure what matters to you. Just as you'd adjust your hiring criteria for different roles, your evaluation standards should adapt to match your specific needs. When you're evaluating technical docs, accuracy might be your top priority. For customer service responses, empathy might matter more. Look at the code snippet below (**Fig 5.3**) to understand this better.



```
"""Evaluate this response focusing on these objectives in order of priority:

{obj_list}

Response to evaluate:
"{response}"

For each objective:
1. Rate achievement (0-{scale}) with specific examples
2. Identify what works well
3. Suggest concrete improvements

Provide a final weighted score (0-{scale}) that reflects the priority weights above.
Include brief justification for the final score."""
```

Fig 5.3: Tuning the evaluation standards to your objectives

5. Utilize Few-Shot Learning

Want your LLM judge to understand exactly what you're looking for? Show it some examples, them clear examples of what "good" looks like, and they'll pick up the pattern quickly.

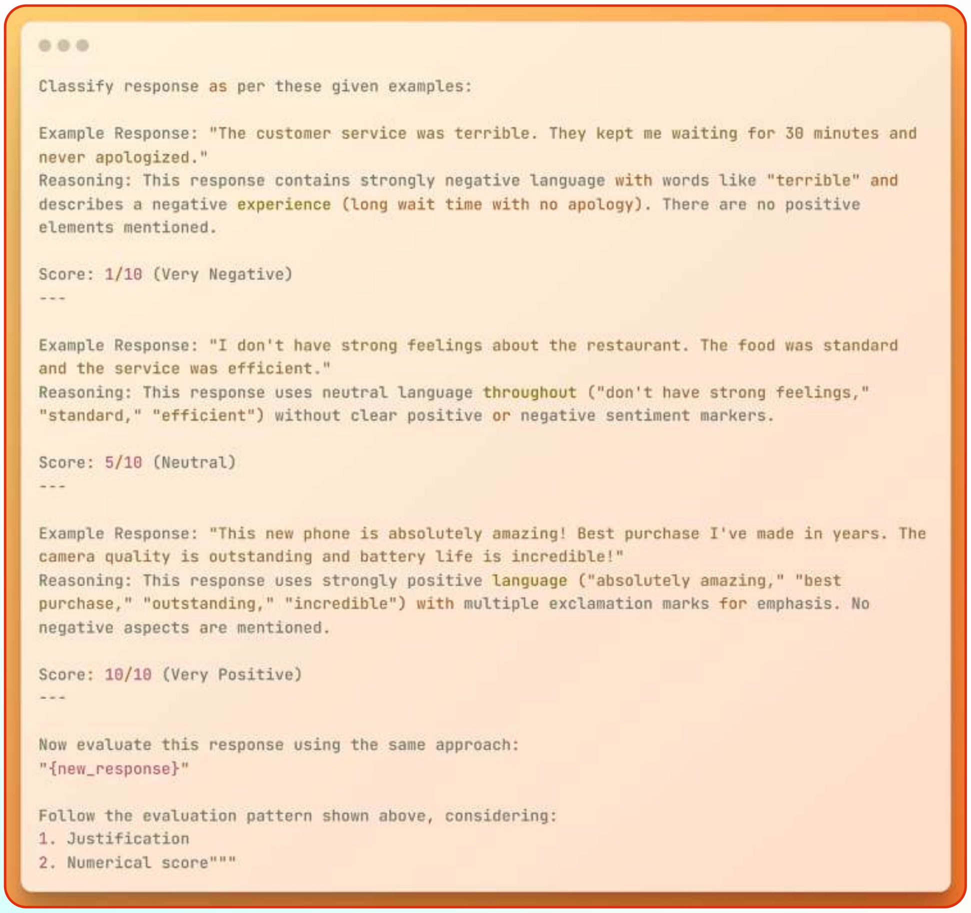
The key is choosing the right examples. You'll want to include a range of responses: excellent ones that nail every criterion, good ones with minor flaws, and weaker ones that need significant improvement. This helps your LLM judge calibrate its evaluations effectively.

Here are a few tips for selecting examples:

- Keep them relevant to your domain
- Include different types of strengths and weaknesses

- Show how specific aspects affect the overall score
- Demonstrate the reasoning behind each evaluation
- Use consistent scoring patterns across examples

Your examples set the standard for evaluation. The better they represent your expectations, the more accurate your LLM judge will be at assessing new responses. You'll see this in the code snippet below. (**Fig 5.4**)



```
Classify response as per these given examples:

Example Response: "The customer service was terrible. They kept me waiting for 30 minutes and
never apologized."
Reasoning: This response contains strongly negative language with words like "terrible" and
describes a negative experience (long wait time with no apology). There are no positive
elements mentioned.

Score: 1/10 (Very Negative)
---

Example Response: "I don't have strong feelings about the restaurant. The food was standard
and the service was efficient."
Reasoning: This response uses neutral language throughout ("don't have strong feelings,"
"standard," "efficient") without clear positive or negative sentiment markers.

Score: 5/10 (Neutral)
---

Example Response: "This new phone is absolutely amazing! Best purchase I've made in years. The
camera quality is outstanding and battery life is incredible!"
Reasoning: This response uses strongly positive language ("absolutely amazing," "best
purchase," "outstanding," "incredible") with multiple exclamation marks for emphasis. No
negative aspects are mentioned.

Score: 10/10 (Very Positive)
---

Now evaluate this response using the same approach:
"{new_response}"

Follow the evaluation pattern shown above, considering:
1. Justification
2. Numerical score"""
```

Fig 5.4: Few-shot prompts

6. Incorporate Adversarial Testing

Your LLM judge needs to handle tricky cases well. Test it with challenging responses - by mixing excellent points with errors, try unusual formats, or include subtle mistakes. These stress tests help you spot where your evaluation system might stumble.

Strong adversarial testing should challenge your evaluation system in several ways (**Table 5.1**): Challenging the evaluation system through adversarial testing.

Content Challenges	Format Challenges	Edge Cases
Technically correct but poorly explained answers	Responses with unusual formatting or structure	Extremely short but accurate responses
Well-written responses with subtle factual errors	Mixing formal and informal language	Very lengthy, detailed responses that miss the point
Responses that sound authoritative but miss key points	Including irrelevant but impressive-sounding details	Answers that challenge common assumptions
Answers that address only part of the question brilliantly	Using uncommon examples or analogies	Responses that require domain expertise to evaluate

7. Implement Iterative Refinement

Your evaluation system should get better over time. Each evaluation offers insights into what works and what needs adjustment. Here's how you can make this improvement cycle work:

What to Track:

1. **Scoring Inconsistencies:**

   When similar responses receive different scores, investigate why. Look for patterns in these differences. Are they happening with particular types of content or specific criteria?

2. **Missed Evaluation Criteria:**

   Keep an eye on which aspects evaluators frequently overlook. This often reveals:

   - Criteria that are too vaguely defined
   - Points that are difficult to assess objectively
   - Areas where evaluators need more guidance

3. **Challenging Response Types:**

   Document responses that consistently cause scoring uncertainty, such as:
   - Highly technical explanations
   - Unconventional approaches to problems
   - Responses that mix strong and weak elements

4. **Conflicting Criteria:**

   Note when different evaluation points seem to work against each other. For instance:
   - Detail vs. conciseness
   - Technical accuracy vs. accessibility
   - Creativity vs. adherence to standards

Where to Make Improvements:

1. **Prompt Structure**
   - Clarify instructions where confusion occurs
   - Add specific examples for challenging cases
   - Remove or rephrase ambiguous language

2. **Evaluation Criteria**
   - Sharpen definitions of vague criteria
   - Add measurement guidelines for subjective aspects
   - Update criteria weights based on importance

3. **Example Sets**
   - Include examples that address common edge cases
   - Update with current, relevant scenarios
   - Show clear distinctions between score levels

After exploring each strategy in detail, let's take a step back and look at the bigger picture. Think of these seven strategies as your toolbox for building a better LLM judge. Some tools help fix common problems, while others help the LLM judge learn and improve. **Table 5.2** brings it all together (**Table 5.2**):

Strategy	Key Benefit	Implementation Focus
Bias Mitigation	Balanced Evaluation	Multiple LLM assessments and result aggregation
Enforced Reasoning	Transparent Decisions	Chain-of-thought prompting with self-reflection
Component Scoring	Precise Assessment	Breaking evaluation into weighted criteria
User Alignment	Targeted Results	Customizing criteria to specific objectives
Few-Shot Learning	Consistent Standards	Strategic example selection and presentation
Adversarial Testing	Robust Evaluation	Challenging edge cases and tricky inputs
Iterative Refinement	Continuous Improvement	Systematic tracking and updates

Table 5.2: Summary of the seven tips to improve your LLM-as-a-Judge

Making a good LLM judge is like training a fair and thoughtful evaluator. These seven tricks help you build one that explains its decisions clearly, stays consistent, and keeps getting better at its job. Start simple, try these improvements one by one, and watch how your evaluations improve.

The Future of AI Evaluation

As artificial intelligence continues to transform businesses across every sector, the challenge of ensuring these systems perform reliably, accurately, and ethically has never been more critical. Throughout this ebook, we've explored how LLM-as-a-Judge evaluation provides a scalable, efficient solution to this growing challenge.

Key Takeaways

We began by examining why traditional evaluation methods fall short when applied to modern language models, particularly at scale. We've seen how LLM-as-a-Judge approaches can overcome these limitations while addressing potential biases and challenges inherent in the evaluation process.

We've explored specialized evaluation models like SLMs that offer cost-effective alternatives without sacrificing quality. We've provided practical frameworks, code examples, and implementation strategies to help you build robust evaluation systems tailored to your specific needs.

Most importantly, we've shared proven techniques—from mitigating biases to implementing iterative refinement—that can significantly enhance your evaluation process and ensure your AI systems meet the highest standards.

The Path Forward

As LLMs become increasingly integrated into critical business functions, the ability to evaluate them effectively will be a key differentiator between organizations that merely deploy AI and those that truly harness its transformative potential.

The frameworks and techniques outlined in this ebook aren't just theoretical concepts—they're battle-tested approaches that have helped leading organizations build more reliable, transparent, and effective AI systems.

Your Next Steps

The time to implement robust evaluation systems is now, before scale makes human review impossible and before any potential issues affect your users or business outcomes.

1. **Assess your current evaluation approach.** Is it scalable? Does it capture all dimensions of quality that matter to your users?
2. **Start small.** Implement LLM-as-a-Judge for a specific use case, using the code examples and frameworks we've provided.
3. **Iterate and refine.** Use the techniques in Chapter 5 to continuously improve your evaluation system.
4. **Build evaluation into your development lifecycle.** Don't treat it as an afterthought—make it central to how you develop and deploy AI systems.

Connect With Us

Ready to take your AI evaluation to the next level? Our team has helped organizations across industries implement effective LLM-as-a-Judge systems that have dramatically improved their AI applications.

Visit galileo.ai to find additional resources, including evaluation examples, benchmark datasets, and case studies from organizations that have successfully implemented these approaches.

Or try our product by signing up for galileo.ai/sign-up

The future of AI belongs to those who can not only build powerful models but also evaluate and improve them effectively. We hope this ebook has equipped you with the knowledge and tools to lead the future with reliable AI.

Glossary

Term	Description
Adherence Score	A metric measuring how well an AI response aligns with and is supported by the provided context.
Attention Bias	A U-shaped attention pattern in LLMs where they give disproportionate weight to information at the beginning and end of texts while potentially missing crucial details in the middle.
Authority Bias	A tendency where LLMs attribute greater credibility to statements from perceived authorities, regardless of the evidence presented.
AUROC	Area Under the Receiver Operating Characteristic curve, a metric that helps evaluate how well a judge balances between being too strict and too lenient.
Beauty Bias	The tendency of LLMs to favor aesthetically pleasing or poetically written content over plain but accurate information.
Chain-of-Thought	A prompting technique that encourages LLMs to break down their reasoning process into sequential steps before providing a final answer.
ChainPoll	A method combining Chain-of-Thought prompting with polling multiple LLM responses to ensure robust and nuanced assessment.
Cohen's Kappa	A statistical measure for evaluating agreement between LLM judges and human evaluators while accounting for chance agreement.
Context Adherence	The degree to which an AI-generated response stays true to and is supported by the provided reference material.
Dynamic Windowing	A technique that splits input context and responses separately to enable comprehensive validation of all context-response pairs.
Evaluation Foundation Models (EFMs)	Specialized models designed specifically for evaluating AI outputs, optimized for assessment tasks while maintaining minimal computational overhead.
Fallacy Oversight Bias	The tendency of LLMs to miss and inadvertently perpetuate logical fallacies in their evaluations.
Few-Shot Learning	A technique where example evaluations are provided to help calibrate an LLM judge's assessment criteria and standards.
Intelligent Chunking	An approach to processing long inputs by splitting them into related segments while maintaining context connections.

LLM-as-a-Judge	An evaluation approach that uses Large Language Models to assess various components of AI systems, including outputs from other models or human annotations.
Luna	A family of Evaluation Foundation Models built on DeBERTa-large architecture, specifically fine-tuned for evaluation tasks.
Multi-headed Design	An architectural approach where models are tailored with specific evaluation criteria in mind, focusing on particular aspects of assessment.
Multi-task Training	A training approach where models learn to conduct multiple types of evaluations simultaneously, enabling shared insights across tasks.
Nepotism Bias	A phenomenon where an LLM evaluator shows favoritism toward content generated by its own model family.
Pairwise Comparison	An evaluation method where two outputs are directly compared to determine which is superior based on specified criteria.
Positional Bias	A systematic bias where LLMs give different scores to the same responses based solely on the order in which they are presented.
RAGTruth	A curated collection of RAG examples spanning multiple categories used for testing and validating evaluation models.
Self-consistency	A concept where valid arguments tend to converge toward correct answers while invalid arguments scatter.
Single Output Scoring	An evaluation method where individual outputs are scored based on predefined criteria, either with or without reference material.
Specialized Language Model (SLM)	Purpose-built small language models designed to excel at specific evaluation tasks while maintaining minimal computational overhead.
Token Level Evaluation	A granular assessment approach that classifies individual sentences or tokens as adherent or non-adherent to the context.
True Positive Rate	A metric calculated as the ratio of correctly identified positive cases to all actual positive cases in evaluation.
Utilization Score	A metric measuring how effectively an AI response uses the available context or reference material.
Verbosity Bias	The tendency of LLMs to equate quantity of information with quality, potentially favoring longer responses over concise, accurate ones.
Zero-shot Evaluation	An evaluation approach where the model assesses outputs without being provided specific examples or training for that particular task.