Evaluating Large Language Models —Principles, Approaches, and Applications

Presented in the NeurIPS Tutorial

Authors: Irina Sigler, Yuan (Emily) Xue

Google

Acknowledgements

Contributors

Bo Li

Ivan Nardini

Anant Nawalgaria

Aida Nematzadeh

Olivia Wiles

Ira Ktena

Designer

Michael Lanning

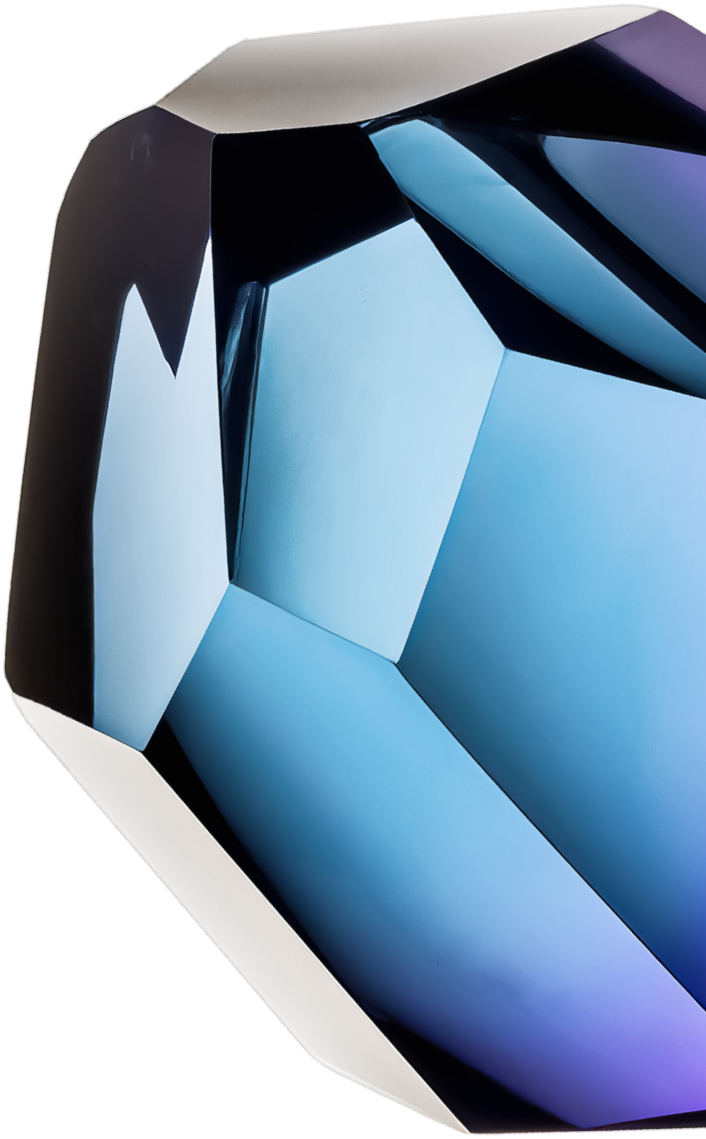


Table of contents

Abstract 5

Evaluation Principles 6

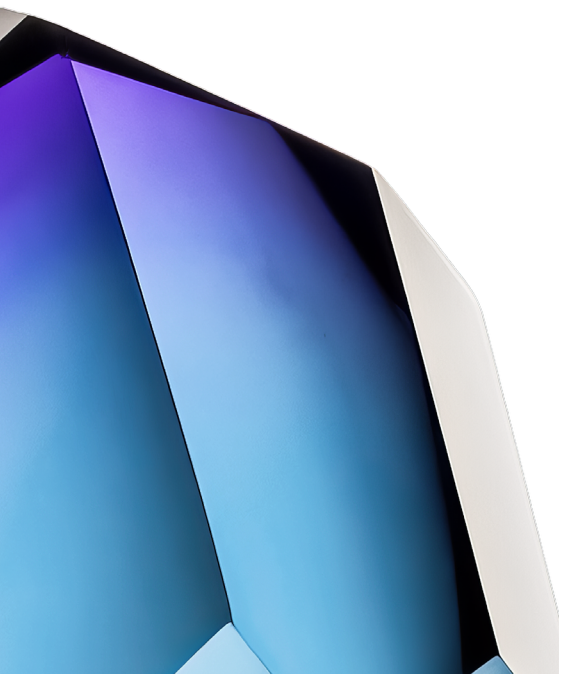
- Evaluation data 7
- Architectural Context 8
- Definition of “Good” 8
- Summary 10

Evaluation Concepts 11

- Point-wise vs Pair-wise Evaluation 11
- Evaluation Criteria 12
- Evaluation Result 13
- Reference Data 14

Evaluation Methods 16

- Overview 16
- Computation-Based Methods 16
- Human Evaluation 18



Autoraters (LLM-based Evaluation) 20
Introduction 20
How to use and design 21
Meta-Evaluation 24
Customization for Your Task 26
Limitations and Mitigations 30
Summary 32
Endnotes 33

Effective LLM evaluation requires a nuanced approach that goes beyond traditional machine learning paradigms.

Abstract

The rapid advancement of Large Language Models (LLMs) has revolutionized various fields, yet their deployment presents unique evaluation challenges. This whitepaper details the principles, approaches, and applications of evaluating LLMs, focusing on how to move from a Minimum Viable Product (MVP) to production-ready systems. It addresses the need for task-specific evaluations, the complexities of evaluating LLM-powered architectures, and the importance of defining "good" outputs in the context of generative models. The whitepaper provides concrete methodologies for automated evaluation, including a practical notebook demonstrating the use of LLMs as autoraters. Finally, we highlight the critical role of meta-evaluation in ensuring the reliability and validity of LLM evaluation systems.

Evaluation Principles

The advent of large language models (LLMs) has dramatically lowered the barrier to entry for building AI-powered applications. Pre-built models eliminate the need for extensive data collection and training, enavbling rapid prototyping and MVP development. It has never been easier to build AI-powered prototypes. However, moving from MVP to production introduces new challenges. Developers face critical decisions regarding prompt engineering, model selection, data augmentation, and ongoing performance monitoring.

A comprehensive evaluation framework is essential for navigating the complexities of LLM application development. Like a compass guiding exploration, it provides a structured approach to making critical decisions and ensuring successful deployment.

This framework helps to:

- Validate application functionality and user experience.
- Identify and mitigate potential issues early on.
- Facilitate clear communication about application capabilities.
- Establish a roadmap for ongoing improvement.

Effective LLM evaluation requires a nuanced approach that goes beyond traditional machine learning paradigms. Three key pillars underpin this approach:

Evaluation data

Generative models possess a remarkable ability to tackle a wide range of tasks. This capability is often showcased through public leaderboards that evaluate performance on standardized tests, each focusing on a specific set of tasks or knowledge areas. While these leaderboards can offer valuable insights for those building and comparing models, they fall short for application developers who require a more tailored approach. Their primary concern lies in understanding how a model performs within the specific context of their application and its anticipated usage patterns. This necessitates the creation of a dedicated evaluation dataset that mirrors the expected production traffic.

Think of this dataset as the foundation upon which your evaluation efforts are built. If it doesn't reflect your real-world usage, even the most sophisticated evaluation methods will prove ineffective. It's like having a compass that doesn't point north – you'll end up confidently navigating in the wrong direction.

To build a robust evaluation dataset:

- Start with a solid foundation: Manually craft initial data points, drawing on expertise from domain experts to ensure relevance and diversity.
- Embrace continuous improvement: Treat your dataset as a living collection, constantly enriching it with real user interactions and production logs as your application evolves
- Explore synthetic data: Leverage the power of LLMs to generate synthetic data, allowing you to test specific scenarios and edge cases that might be underrepresented in your real-world data.

Architectural Context

Evaluating LLMs requires more than just examining the final output. While your initial MVP might involve a simple input/output flow, production-ready LLM applications often integrate the model within a complex architecture. Think of it as an orchestra, with the LLM playing a key role, but not performing solo.

These architectures might incorporate:

- **Data Augmentation**: Techniques like Retrieval Augmented Generation (RAG) enrich the LLM's knowledge with external information sources. Evaluating the effectiveness of this augmentation is crucial.
- **Agentic Workflows**: LLMs can be used to build complex agent systems capable of taking a series of actions to achieve a goal. Evaluation needs to consider the entire journey – the agent's ability to plan, reason, make decisions, take actions, and reflect on outcomes – not just the final destination.

Therefore, your evaluation process should shift from assessing just the model to analyzing the entire system. This approach ensures you understand how all components interact and contribute to the application's overall performance.

Definition of "Good"

When evaluating traditional machine learning models, we typically compare predictions against a known "ground truth." However, the creative nature of generative AI means there can be multiple valid solutions to a single problem. For example, even if I craft what I believe to be a perfect summary of an annual report, a generative AI model might surprise me with an equally insightful and well-written alternative.

Traditional metrics that prioritize matching a single "correct" answer would unfairly penalize these unexpected solutions. To truly harness the power of generative AI, we need evaluation methods that embrace this creativity, rather than limiting the technology's potential by forcing it to conform to a single predefined output.

To ensure evaluation metrics are aligned with your business outcome, establishing precise criteria that reflect those desired outcomes is crucial during the development process. Think of it as creating a wishlist of desired attributes. For example, you might prioritize "simplicity," meaning the output should use clear and concise language. As you gather evaluation data and user feedback, refine your criteria to be more specific to your use case. Instead of just stating a standard definition of "simplicity," provide concrete examples and explain how you'll assess it. This also explains why pre-built criteria can be a useful starting point, but most often will require adjustments to fit a specific use case.

For complex tasks consider using LLMs for helping to refine these criteria into rubrics with specific questions for each data point to guide evaluation. This adds another layer of granularity and ensures a thorough assessment.

One advantage of this approach compared to "traditional" machine learning evaluation? Your evaluation criteria, written in plain language, become a shared definition of "good" that everyone on your team can understand and contribute to. This fosters collaboration and avoids the confusion often associated with explaining complex machine learning metrics.

Summary

To sum up, evaluating LLM-powered applications requires a tailored approach:

- **Data is the Foundation**: Create a dedicated dataset that mirrors your actual application usage. Continuously refine this dataset with real-world data and user feedback. Remember, garbage in, garbage out – your evaluation is only as good as the data it's based on.
- **Look Beyond the Model**: LLMs don't operate in isolation. Evaluate your entire application architecture, including prompt engineering, data augmentation, and any other components. Analyze intermediate steps and overall workflow, not just the final output.
- **Define "Good" Precisely**: Clearly articulate what constitutes a successful outcome for your specific use case. The more specific your criteria, the more effective your evaluation will be. Don't just rely on general metrics – define what matters most for your application.

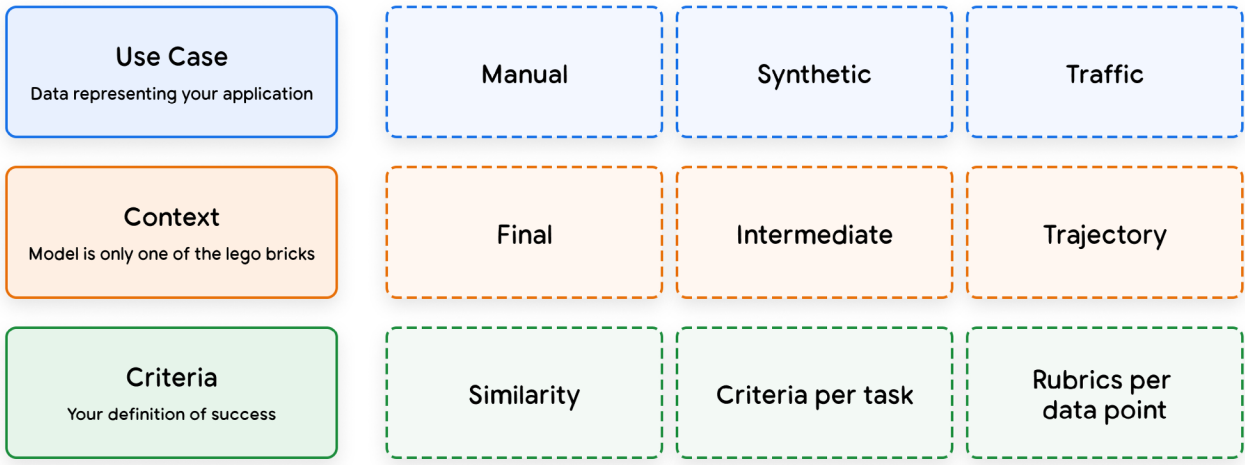


Figure 1: Overview task-specific evaluation

Evaluation Concepts

When evaluating your LLM application, your north star should be a metric that directly reflects your desired business outcome. At its core, evaluation can be represented as a function F with two key parameters (subject, criteria):

```
F(subject, criteria) → result
```

In this function, the `subject` is the thing you're evaluating (e.g., an LLM's output, a specific component of your application), while the `criteria` represent the lens through which you're evaluating (e.g., accuracy, relevance, fairness). The `result` is the qualitative assessment of your system's quality.

Point-wise vs Pair-wise Evaluation

Let's delve deeper into the "subject" of LLM evaluation. There are two primary paradigms to consider as shown in Image 2:

1. **Point-wise Evaluation**: This paradigm assesses the absolute quality of the output, independent of any other responses.
2. **Pair-wise Evaluation**: Also known as side-by-side evaluation, this paradigm compares two different outputs generated for the same input. One output typically serves as a baseline (e.g., from a previous model version), while the other is the target output you're evaluating. This head-to-head comparison determines which output is superior for the given input.

While image 2 depicts a simplified scenario evaluating a single model, the actual scope of evaluation can be much wider. Rather than assessing the LLM in isolation, consider evaluating your complete production system.

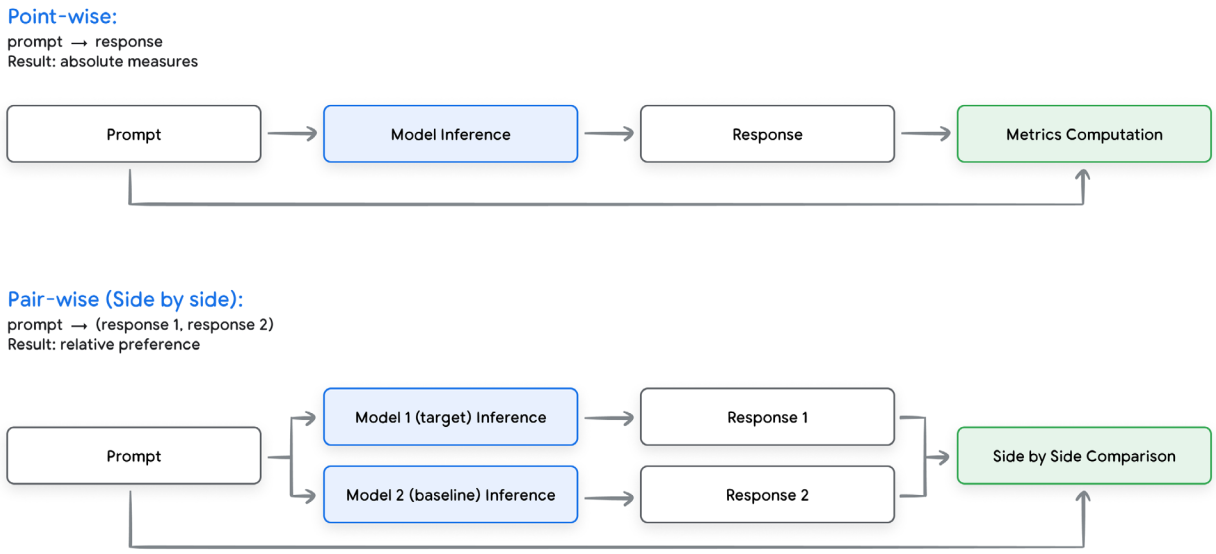


Figure 2: Point-wise and pair-wise evaluation

Evaluation Criteria

Let's dissect the "criteria" component of our evaluation function. This involves two key elements:

1. **Defining Dimensions**: Dimensions represent the specific qualities or aspects you want to evaluate. Many dimensions are task-specific. For general text generation, you might prioritize fluency and coherence. For summarization, conciseness and comprehensiveness become more relevant. Other tasks, like open-book question answering, might require

checks for groundedness (i.e., how well the answer is supported by the provided source). These standard dimensions might require adjustments depending on your context. For example, "fluency" in a research report will differ significantly from "fluency" in the context of a retail assistant chatbot. Beyond these common dimensions, you might need to add bespoke criteria tailored to your specific use case. Perhaps you need to assess how entertaining, engaging, or intuitive an LLM's output is. Interestingly, even with generative models, some evaluation tasks can be framed as discriminative tasks. For instance, evaluating code generation often boils down to checking the correctness of the execution result – a binary classification of "correct" or "incorrect." In agentic scenarios, you might focus on dimensions like tool selection accuracy and parameter value correctness.

2. **Establishing Rubrics**: Once you've defined your dimensions, you need rubrics to quantify them. Rubrics provide a framework for assigning scores to different levels of performance. For example, a rubric for "coherence" might include a scale ranging from "incoherent and nonsensical" to "perfectly clear and logically structured," with corresponding numerical scores for each level.

Evaluation Result

Let's wrap up our exploration of evaluation criteria by examining the results themselves. Evaluation results typically consist of two key components:

1. **Ratings**: Ratings provide a quantitative measure of performance. In point-wise evaluation, this is an absolute measure of quality (e.g., scoring an output on a scale of 1 to 5). In pair-wise evaluation, the rating reflects a relative preference, indicating which output is superior (see figure 3 below).

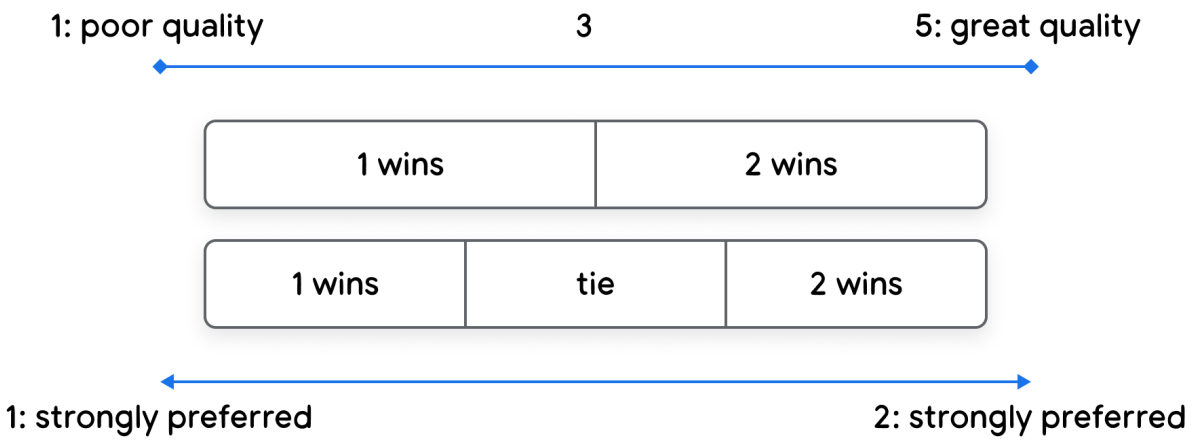


Figure 3: Evaluation results

2. **Rationales**: Rationales offer qualitative insights in the form of verbal feedback. They explain why a particular rating was assigned. This is valuable to understand the rater's reasoning process, which can be useful in identifying areas for improvement. To illustrate, a rater might mention a dimension that is not relevant to the use case or shows to misinterpret the rubrics. The act of providing a rationale encourages evaluators to think critically and justify their assessments, leading to more thoughtful and consistent evaluations and shown to enhance the overall evaluation quality.

Reference Data

Let's discuss the role of reference data in LLM evaluation. While it's a cornerstone for evaluating discriminative models, its role in generative AI is more nuanced.

In discriminative models, reference data provides the ground truth against which predictions are compared. However, the open-ended nature of generative tasks often means there isn't a single "correct" answer. Instead, a good reference might be a representative sample from the distribution of possible valid outputs.

Consider this analogy: imagine evaluating a chef. For a discriminative task like replicating a classic dish, a detailed recipe (the reference data) provides a clear benchmark for comparison. But for a generative task like creating a new dessert, there's no single "correct" answer. Instead, you might evaluate the chef's creation based on criteria like taste, originality, and presentation, potentially comparing it to a range of existing desserts (the reference data) for inspiration and context.

Figure 4 illustrates how the evaluation setup changes when shifting from predictive to generative tasks, highlighting the evolving role of reference data.

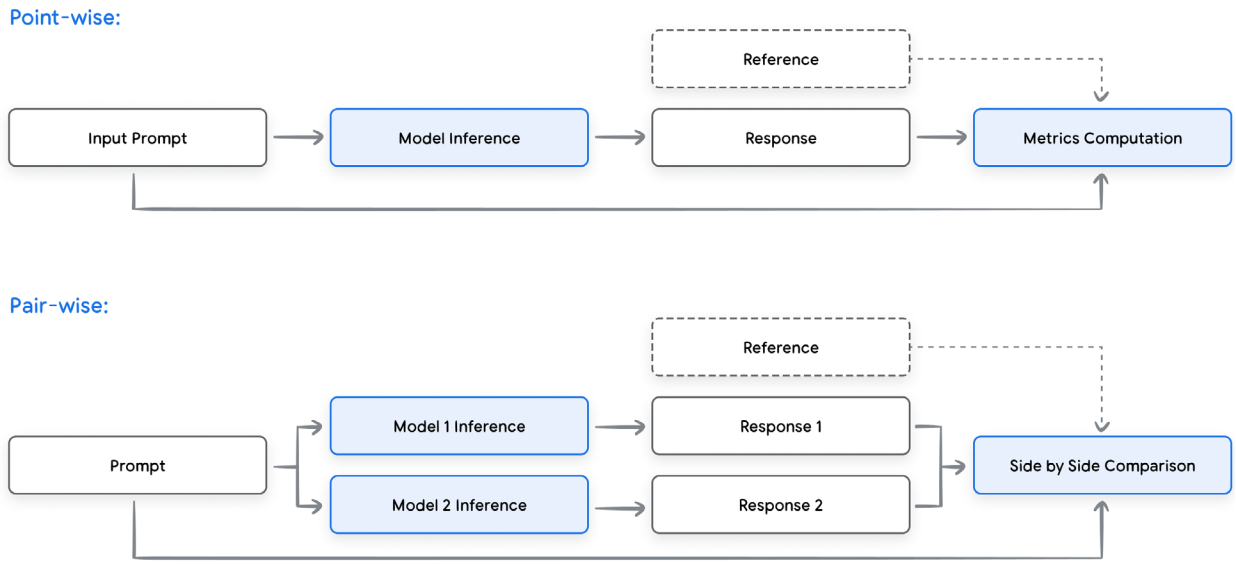


Figure 4: Point-wise and pairwise with reference data

Evaluation Methods

Overview

To effectively measure the impact of LLMs on business outcomes, we need to identify reliable surrogate metrics for your business outcomes. We can group the existing evaluation methods into three categories that we are going to explore in more depth in this chapter:

1. **Computation-Based Methods**: These methods leverage quantitative measures to provide objective insights into LLM performance. Examples include BLEU scores, ROUGE scores, and perplexity.
2. **Human Evaluation**: Human judgment is considered the gold standard for assessing nuanced aspects of generative tasks. This can include human raters, user studies, expert reviews, and A/B testing.
3. **Autoraters**: LLM-powered autoraters offer scalability and efficiency in evaluation. These models can be calibrated to approximate human judgments on specific tasks.

Computation-Based Methods

Computation-based methods offer a quantitative approach to evaluating LLMs by measuring the similarity between a model's output and a given reference. These methods, while efficient and objective, have inherent limitations. They only support pointwise evaluation, resulting in a single score. While you can compare and rank the scores for any number of outputs, it is important to keep in mind that these metrics measure each response in isolation. While you can customize the formula for any computation-based metric, you cannot incorporate fine-grained criteria or nuanced aspects.

Two prominent approaches within computation-based methods include:

1. **Lexical Similarity**: These methods, exemplified by ROUGE and BLEU, focus on syntactic similarities by analyzing word frequencies, phrases, and n-grams. Two examples:
    - ROUGE (Recall-Oriented Understudy for Gisting Evaluation): A suite of metrics with variations like ROUGE-n (examining n-grams) and ROUGE-L (based on the longest common subsequence). ROUGE scores range from 0 to 1, indicating the degree of overlap between the generated text and the reference.
    - BLEU (Bilingual Evaluation Understudy): Originally designed for machine translation evaluation, BLEU measures the precision of n-grams in the generated text compared to the reference.
2. **Embedding Similarity**: Methods like BERTScore and BARTScore delve into the semantic similarity of responses by comparing the contextualized embeddings of words and phrases. This allows for a deeper understanding of meaning beyond surface-level matches.

It is crucial to acknowledge that computation-based methods are sensitive to the choice of reference text and often exhibit a weak correlation with human judgments, particularly for complex, open-ended tasks. However, they remain valuable for:

- **Scalable Evaluation**: Providing rapid and efficient assessment in simple, well-defined settings.
- **Component-Specific Evaluation**: Breaking down complex tasks into smaller components where objective answers are required, such as in function calling or parameter value comparison.

- **Model Tuning and Monitoring**: Offering a low-cost sanity check and tracking progress during model training by measuring the convergence of generated output towards desired references.

When employing computation-based methods, consider these best practices:

- **Careful Reference Selection**: Thoughtfully curate reference texts to ensure they accurately represent the desired output. Multiple references can be used to increase the robustness of the evaluation.
- **Preprocessing**: Remove noise and irrelevant information (e.g., punctuation, stop words) to enhance the accuracy of the evaluation.
- **Complementary Use**: Combine computation-based methods with human evaluation and autoraters to gain a holistic understanding of LLM performance.

By understanding the strengths and limitations of computation-based methods, and by adhering to best practices, developers can leverage these techniques effectively to gain valuable insights into LLM performance and drive improvements in their models.

Human Evaluation

In many teams, human evaluation remains indispensable in LLM development, providing qualitative insights and ensuring alignment with human values and preferences. While resource-intensive, a structured approach of working with human evaluators maximizes its effectiveness and minimizes costs.

You need to start with a pilot study aiming to calibrate the human raters. Begin with a small sample of LLM outputs that you provide for evaluation. Develop a clear and detailed rubric outlining evaluation criteria and desired qualities. Crucially, calibrate the inter-rater alignment by having multiple raters assess the same samples and review discrepancies. Asking raters to provide a rationale for their assessment might help you identify flaws in your evaluation rubric and confirm accurate interpretation of the criteria. Keep on iterating until you achieve good alignment amongst the raters and with your criteria.

Once the rubric and rater understanding are aligned, expand the evaluation to a larger, representative dataset. Still, working with raters does not end here. Treat it as an iterative process allowing for ongoing refinement of both the rubric and the evaluation process itself.

While invaluable, human evaluation has limitations:

- **Cost and Time**: Large-scale human evaluation can be expensive and time-consuming.
- **Expertise Requirements**: Tasks requiring specialized domain knowledge or linguistic expertise necessitate carefully vetted rater services.
- **Bias**: Human evaluators are inherently susceptible to biases, which can influence their judgments.

To mitigate these challenges you might consider professional annotation services or domain experts for specialized tasks. Additionally, you can implement strategies to reduce bias, such as anonymizing outputs, diversifying the pool of raters, and providing comprehensive training.

Human evaluation is most critical in two steps of your development process:

- **Production Release**: Directly inform decision-making for product readiness by rigorously assessing whether quality standards meet production requirements.

- **Autorater Calibration and Optimization**: Generate a smaller, high-quality human-labeled dataset to assess and iteratively improve the performance of LLM-based autoraters, enabling scalable evaluation.

By strategically incorporating human evaluation, developers can ensure their LLMs meet the highest quality standards, align with user expectations, and ultimately drive successful real-world applications.

Autoraters (LLM-based Evaluation)

Introduction

Autoraters, also known as LLM-based evaluators, leverage the capabilities of LLMs as judges. They offer scalability and efficiency while retaining the flexibility to address diverse evaluation needs. A key advantage compared to human raters is that autoraters enable the efficient evaluation of large volumes of LLM outputs, making them particularly well-suited for applications with high throughput requirements. Like human evaluation, autoraters support both pointwise and pairwise comparisons, allowing for tailoring the evaluation criteria to specific use cases. Unlike traditional computation-based methods, autoraters can be designed to operate with or without reference data, broadening their applicability to open-ended and creative tasks. Finally, LLM-based autoraters can generate human-readable rationales for their judgments, providing valuable insights into the evaluation process.

How to use and design

In the most simple setup you provide the task, including criteria and candidate responses (and optionally, a reference). The task is presented to the autorater, which formats it into a prompt for the autorater LLM model, and parses the model output to create the final evaluation results (see figure below).

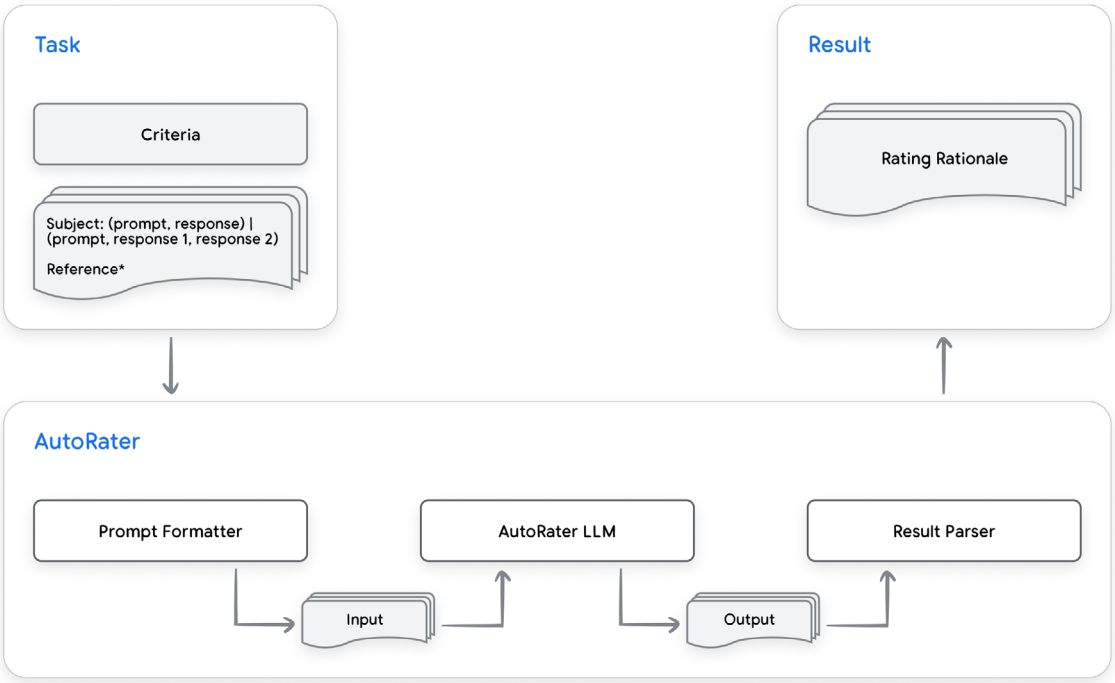


Figure 5: Basic design of autoraters

Let's start by looking at the types of models that can serve as autoraters.

- **Generative Models**, including foundation and fine-tuned autoraters, leverage language generation to provide scores, detailed rationales with insights into their decision-making process. These models support flexible output formatting, including pointwise scoring and pairwise comparisons. While offering many advantages, they require result parsing, which can be prone to errors.
- **Discriminative Models** (also known as Reward Models). These models are trained to predict scalar scores and are optimized for precise and consistent evaluations based on predefined criteria. They support both pointwise scoring and pairwise comparisons, but unlike Generative Models, they do not provide rationales or nuanced reasoning.
- **Implicit Reward Models** trained via Direct Preference Optimization (DPO) are less common and generally underperform compared to Discriminative and Generative models, so they are not the primary choices for autorater yet.

RewardBench provides a leaderboard for popular autorater models, ranking them based on their performance across a diverse set of evaluation datasets.

How do autoraters format evaluation tasks into prompts? To ensure accurate and consistent evaluations from the autorater, it's crucial to structure your prompts carefully. A well-formatted prompt consists of three key components as shown in image 6: First, you provide evaluation instructions explaining that the model should act as an autorater. Begin by clearly defining the autorater's role as an expert evaluator. Provide explicit instructions on the task, such as: "Your task is to evaluate the quality of the responses generated by AI models based on the following criteria...". Next you outline the specific criteria for evaluation. Clearly define each criterion and provide concise explanations. Keep in mind that performance improves as you provide more specific criteria (see Kim et al. 2024, for details on specific criteria & Shankar et al. 2024 for iterative criteria refinement). For each criterion, establish a clear rating scale with corresponding descriptions for each level. Finally, you point to the data that is being evaluated, including the prompt and response of the candidate models,

optionally you can also include references to illustrate the ideal output. For generative autorater models, the prompt also needs to include the output format specification, such as "Your output should be formatted as ..."

How do autoraters turn the judge LLM output into evaluation results? When using a generative autorater model, its text output needs to be parsed according to the specified format. This process should handle potential errors from malformed outputs and produce structured results.

Evaluating LLMs requires tailored methodologies to tackle task-specific challenges, define quality, and bridge the gap from MVPs to production systems. This Colab notebook offers an end-to-end example of the basic design of an autorater.

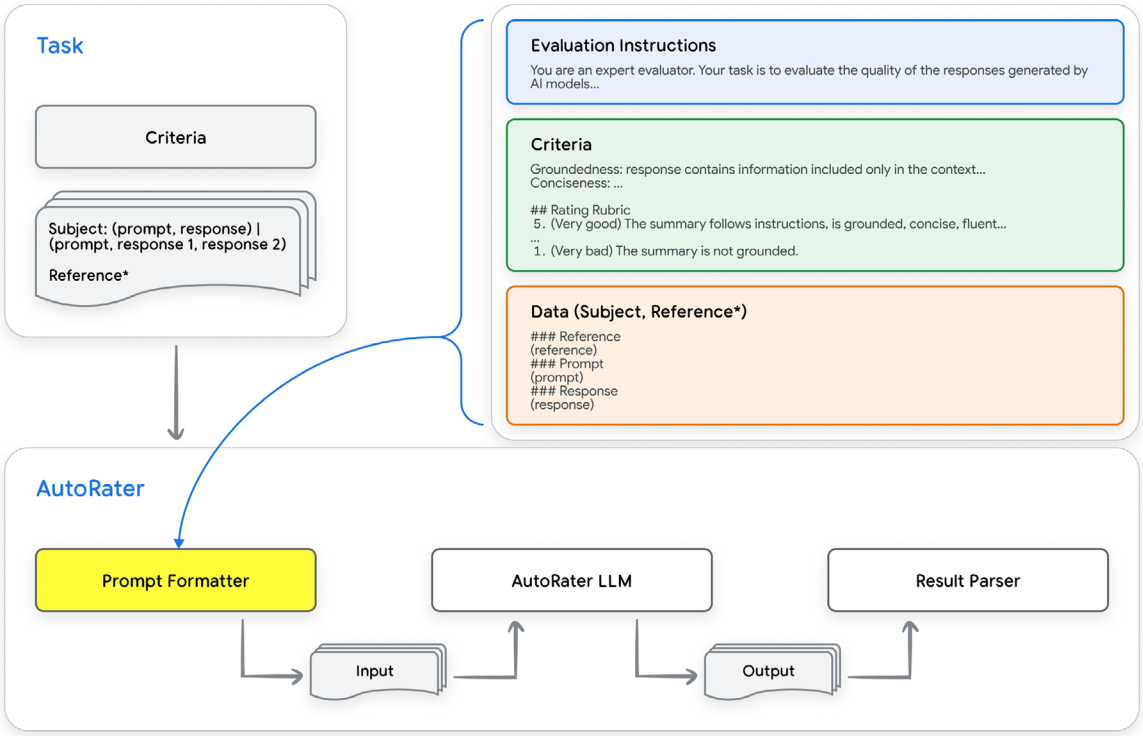


Figure 6: Autorater prompt

Rather than relying solely on a single LLM, an autorater can leverage multiple LLMs from different model families or multiple assessments from the same LLM, described in Juries (Verga 2024), ChatEval (Chan 2023), Agent-as-Judge (Zhuge 2024), MATEval (Li 2024).

Meta-Evaluation

Similarly to how you need to calibrate your compass, it is key to evaluate your evaluator and understand if it is pointing you to the right direction. Thus, meta-evaluation is a critical step in your work to set up a task-specific evaluation framework. This process involves calibrating autoraters against human judgments, essentially evaluating how well these automated systems align with human preferences. This calibration is often achieved through agreement or correlation measures, depending on the nature of the evaluation task.

- **Agreement** for pair-wise comparisons.
  - **Cohen's Kappa**: Measures the agreement between two raters on categorical data, accounting for chance agreement. When calibrating autoraters, it assesses the agreement between human ratings and the autorater's output. Generally, a Kappa value above 0.8 is considered to indicate strong agreement, while a value above 0.6 suggests moderate agreement.
  - **Confusion matrix and classification metrics** (including accuracy, precision, recall, specificity, F1 score): Assesses how well the autorater performs against the gold standard set of human annotations. It shows the autorater's accuracy and error patterns.
- **Correlations** for point-wise scoring.
  - **Spearman correlation**: is good for monotonic relationships and is less sensitive to outliers.

- **Kendall's Tau**: is suitable for ranked data and assessing concordance/discordance and handles ties well.
- **Pearson correlation**: is best for linear relationships with normally distributed data.

To facilitate robust meta-evaluation, a wealth of OSS datasets with human annotations have been created, encompassing diverse usage scenarios and evaluation criteria.

Here is a non-exhaustive list of meta-evaluation datasets available:

- MTBench and Chatbot Arena [pair-wise] Multi-turn conversations, crowdsource preference annotations.
- HelpSteer and HelpSteer2 [pair-wise] helpful, factually correct and coherent, leveraging human annotators.
- LLMBar [pair-wise] manually curated challenging meta-evaluation to assess instruction-following.
- AlpacaEval and AlpacaFarm [pair-wise], chat, low-cost simulation of pairwise feedback from API models.
- Anthropic Helpful and Anthropic HHH [pair-wise]: human alignment capability on helpful, honest, harmless.
- summarize_from_feedback [pair-wise], summary comparison.
- HuanEvalPack [point-wise] coding abilities.
- FLASK [point-wise]: fine-grained scoring with 4 primary abilities divided into 12 fine-grained skills.

These datasets are often integrated into comprehensive benchmarks that enable researchers to evaluate and compare autoraters:

- RewardBench: [5 category with 27 datasets], comprehensive benchmark that covers chat, reasoning, and safety.
- LLM-AggreFact; [11 datasets] fact verification benchmark covering: fact verification, faithfulness of summary, etc.
- JudgeBench: benchmark on challenging response pairs spanning knowledge, reasoning, math, and coding.
- WildBench: WB-Reward and WB-Score with fine-grained outcomes. e.g. for pairwise comparison: much better, slightly better, slightly worse, much worse, or a tie.
- EvalBiasBench: bias benchmark
- CoBBLEr : bias benchmark

Customization for Your Task

While existing benchmarks offer valuable insights, it's crucial to tailor your evaluation process to your specific needs.

Consider the following key aspects while aligning your meta-evaluation with your intended use case to ensure accurate and relevant results:

1. **Prompt Curation**: The prompts used in your evaluation should closely mirror your intended production usage distribution. Benchmarks like HelpSteer demonstrate this by using crowdsourced prompts to cover diverse use cases. However, for optimal results, curate your own prompt set. This can be done by manually creating prompts or sampling directly from your production traffic to ensure your evaluation is grounded in the specific challenges your model will face.
2. **Candidate Response Generation**: Focus your evaluation on the specific language models you are considering for evaluation and productionisation. While benchmarks like MT-Bench and Chatbot Arena include a wide range of models, your meta-evaluation should directly compare the performance of your candidate models. This targeted approach provides the most relevant insights for your decision-making process..
3. **Annotation Quality**: High-quality human annotations are the gold standard for meta-evaluation. While potentially expensive, human judgment provides the most reliable ground truth for assessing your autorater's performance. To ensure consistency, pay close attention to inter-rater agreement. For more scalable and efficient options, consider using powerful language models as annotators, but proceed cautiously and be mindful of potential biases, such as self-promotion.

Meta-evaluation provides valuable insights into the effectiveness of your autorater, but it's not the end of the process. Once you have a clear understanding of your autorater's strengths and weaknesses, you can leverage various strategies to improve its performance.

These strategies include:

- Adjusting Evaluation Criteria: Revisit your evaluation criteria to ensure they accurately reflect the desired qualities of the output. You might need to refine existing criteria or introduce new ones to capture nuances specific to your task.

- Incorporating Diverse Judges: Expand the pool of judges to include a wider range of perspectives and expertise. This can help mitigate biases and improve the robustness of your evaluation.
- Fine-tuning the Judge Model: If you're using a language model as your autorater, fine-tune its parameters to align more closely with human judgments on your specific task. This often involves training the model on a dataset of human-annotated examples.

When fine-tuning your autorater, remember that base model selection is crucial. Start with a strong foundation by choosing a pre-trained language model that demonstrates good performance on related tasks. Then, focus on optimizing your prompts and model settings before adjusting the model's weights. The specific training method you employ will depend on the type of model you're using and the nature of your evaluation task.

Here are a few examples to show different training strategies for various base models:

Model	Base Model	Type	Training Data	Training Method
FLAMe-24B	PaLM-2-24B (IT)	generative	100+ quality assessment tasks comprising 5M+ human judgments	Text-to-text multitask SFT
FLAMe-RM-24B; FLAMe-Opt-RM	PaLM-2-24B (IT)	discriminative	HelpSteer, PRM800K, CommitPack, HH Harmlessness (covering chat, reasoning and safety)	Fine-tuning with pairwise preference data Tail-patch fine-tuning to optimize multitask mixture
Skywork-Reward	Gemma-2-27b-it; Llama-3.1-8B	discriminative	Skywork-Reward-Preference-80K-v0.1 (HelpSteer2, OffsetBias, WildGuard, Magpie DPO series, In-house human annotation data)	BT-based pair-wise ranking loss with a few variants and careful curation and filtering of training data
Skywork-Critic	Llama-3.1-8B-Instruct; Llama-3.1-70B-Instruct	generative	Skywork-Reward-Preference-80K-v0.1	Instruction-tuning focusing on pairwise preference evaluation and general chat tasks.
Nemotron-Reward	Llama-3.1-70B-Instruct; Nemotron-4-340B	discriminative	HelpSteer2	Linear layer converts the final layer of the end token into 5 scalar values, train with MSE loss
PROMETHEUS 2	Mistral 7B & 8x7B	discriminative	PREFERENCE COLLECTION (1K score rubrics, 20K instructions & reference answers, 200K responses pairs & feedback)	SFT Joint point-wise and pair-wise training with weight merging to produce final model
InstructScore	Llama-2-7B	generative	10k raw from 100 domains	Multitask SFT over reference output and diagnostic report

Table 1: Tuning autoraters

Limitations and Mitigations

Although autoraters provide a scalable solution for evaluating LLMs, it's crucial to recognize potential limitations and implement appropriate mitigation strategies.

Two primary concerns are:

1. **Biases**: Autoraters, can exhibit various biases that affect their judgments. Common biases include:
    - Position Bias: Favoring responses present in a specific position, e.g., the first or last
    - Verbosity/Length Bias: Preferring longer responses, even if they are not necessarily more accurate or relevant.
    - Self-enhancement/Egocentric Bias: Demonstrating a preference for answers generated by the models of the same model family.
2. **Lack of consistency**: Autoraters may lack consistency in their judgments due to prompt sensitivity or inherent randomness in their outputs.

    Several mitigation strategies can address these limitations:

    - Prompt Engineering and Orchestration
        - Carefully crafting prompts, and providing details on evaluation criteria and rubrics.
        - Swapping Positions: Calling the autorater multiple times with reversed option order can help identify and reduce position bias.
        - Self-Consistency Checks: Evaluating the same input multiple times and analyzing the outputs for consensus can improve consistency and reduce the impact of randomness.

    - Panel of Diverse Models: Employing a "jury" of language models from different families can help mitigate biases associated with individual models.
    - In-context Learning: Providing the autorater with a few examples of good judgments can guide its evaluation process.
- Fine-tuning with De-biasing Datasets: Training the autorater on datasets specifically designed to address biases can help reduce their influence.

For a deeper dive into these limitations and mitigation techniques, refer to the following resources: MT-Bench (Zheng 2023), OffsetBias (Park 2024), CoBBLEr (Koo 2024), Juries (Verga 2024), Length-Controlled AlpacaEval (Dubois 2024), Position Bias (Shi 2024).

Summary

We explored different approaches to evaluating LLMs, emphasizing the need to align evaluation strategies with specific requirements and address potential biases. We examine three primary evaluation methods: computational metrics, human assessment, and automated evaluation using autoraters. The optimal choice depends on the specific task, balancing cost considerations with the desired level of quality. Importantly, these methods are not mutually exclusive; they can be used together to provide a more comprehensive and robust evaluation.

Customization is key to effective LLM evaluation. This includes carefully designing prompts to elicit desired responses, fine-tuning LLMs to better align with human judgments, and calibrating autoraters through meta-evaluation to ensure they accurately reflect your evaluation criteria. Throughout the process, it's vital to remain focused on your specific business needs, ensuring the evaluation aligns with your domain, criteria, and objectives.

Finally, it's crucial to be mindful of potential biases in evaluation methods. By understanding and addressing these biases proactively, you can develop more reliable evaluation strategies that accurately assess LLM performance.

A forthcoming notebook will delve deeper into these concepts, providing practical guidance and examples for implementing robust LLM evaluation techniques.model to tell a story or write a song. Examples of prompt engineering include providing clear instructions to the LLM, giving examples, using keywords, and formatting to emphasize important information, providing additional background details etc.

Endnotes

1. Doddapaneni, S., Khan, M. S. U. R., Verma, S., & Khapra, M. M., 2024, 'Finding Blind Spots in Evaluator LLMs with Interpretable Checklists'. Available at: https://arxiv.org/abs/2406.13439.
2. Li, D., et al., 2024. 'From Generation to Judgment: Opportunities and Challenges of LLM-as-a-judge'. Available at: https://arxiv.org/abs/2411.16594.
3. Zheng, L., et al., 2024, 'Judging LLM-as-a-judge with MT-Bench and Chatbot Arena'. Available at: https://arxiv.org/abs/2306.05685.
4. Kahng, M., et al., 2024, 'LLM Comparator: Visual Analytics for Side-by-Side Evaluation of Large Language Models'. Available at: https://arxiv.org/abs/2402.10524.
5. Dubois, Y., et al., 2024, 'Length-Controlled AlpacaEval: A Simple Way to Debias Automatic Evaluators'. Available at: https://arxiv.org/abs/2404.04475.
6. Lin, B. Y., et al., 2024, 'WildBench: Benchmarking LLMs with Challenging Tasks from Real Users in the Wild'. Available at: https://arxiv.org/abs/2406.04770.
7. Bai, Y., et al., 2023, 'Benchmarking Foundation Models with Language-Model-as-an-Examiner'. Available at: https://arxiv.org/abs/2306.04181.
8. Liu, Y., Moosavi, N. S., & Lin, C., 2023, 'LLMs as Narcissistic Evaluators: When Ego Inflates Evaluation Scores'. Available at: https://arxiv.org/abs/2311.09766.
9. Vu, T., Krishna, K., Alzubi, S., et al., 2024, 'Foundational Autoraters: Taming Large Language Models for Better Automatic Evaluation'. Available at: https://arxiv.org/abs/2407.10817.
10. Lambert, N., et al., 2024, 'RewardBench: Evaluating Reward Models for Language Modeling'. Available at: https://arxiv.org/abs/2403.13787.
11. Xu, W., et al., 2023, 'INSTRUCTSCORE: Explainable Text Generation Evaluation with Finegrained Feedback'. Available at: https://arxiv.org/abs/2305.14282.
12. Kim, S., et al., 2024, 'Prometheus 2: An Open Source Language Model Specialized in Evaluating Other Language Model'. Available at: https://arxiv.org/abs/2405.01535.
13. Zhang, Y., et al., 2023, 'LLMEval: A Preliminary Study on How to Evaluate Large Language Models'. Available at: https://arxiv.org/abs/2312.07398.

14. Wang, Z., et al., 2024, 'HelpSteer2: Open-source dataset for training top-performing reward models'. Available at: https://arxiv.org/abs/2406.08673.

15. Vodrahalli, K., et al., 2024, 'Michelangelo: Long Context Evaluations Beyond Haystacks via Latent Structure Queries'. Available at: https://arxiv.org/abs/2409.12640.

16. Wiles, O., et al., 2024, 'Revisiting Text-to-Image Evaluation with Gecko: On Metrics, Prompts, and Human Ratings'. Available at: https://arxiv.org/abs/2404.16820.

17. Shen, T., et al., 2023, 'Large Language Model Alignment: A Survey'. Available at: https://arxiv.org/abs/2309.15025.

18. Wang, P., et al., 2023, 'Large Language Models are not Fair Evaluators'. Available at: https://arxiv.org/abs/2309.15025.

19. Soni, M., & Wade, V., 2023, ' Comparing abstractive summaries generated by chatgpt to real summaries through blinded reviewers and text classification algorithms'. Available at: https://arxiv.org/abs/2303.17650.

20. Verga, P., et al., 2024, 'Replacing Judges with Juries: Evaluating LLM Generations with a Panel of Diverse Models'. Available at: https://arxiv.org/abs/2404.18796.

21. Zhou, H., et al., 2024, 'Fairer Preferences Elicit Improved Human-Aligned Large Language Model Judgments'. Available at: https://arxiv.org/abs/2406.11370.

22. Liu, Y., et al., 2024, 'Aligning with Human Judgement: The Role of Pairwise Preference in Large Language Model Evaluators'. Available at: https://arxiv.org/abs/2403.16950.

23. Kim, T. S., et al., 2024, 'EvalLM: Interactive Evaluation of Large Language Model Prompts on User-Defined Criteria'. Available at: https://arxiv.org/abs/2309.13633.

24. Shankar, S., et al., 2024, 'Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences'. Available at: https://arxiv.org/abs/2404.12272.

25. Zhu, L., et al., 2023, 'Judgelm: Fine-tuned large language models are scalable judges'. Available at: https://arxiv.org/abs/2310.17631.

26. Wu, T., et al., 2024, 'Meta-rewarding language models: Self-improving alignment with llm-as-a-meta-judge'. Available at: https://arxiv.org/abs/2407.19594.

27. Wang, T., et al., 2024, 'Self-taught evaluators'. Available at: https://arxiv.org/abs/2408.02666.

28. Wang, B., et al., 2024, 'Halu-J: Critique-Based Hallucination Judge'. Available at: https://arxiv.org/abs/2407.12943.

29. Park, J., et al., 2024, 'Offsetbias: Leveraging debiased data for tuning evaluators'. Available at: https://arxiv.org/abs/2407.06551.

30. Xie, T., et al., 2024, 'Sorry-bench: Systematically evaluating large language model safety refusal behaviors'. Available at: https://arxiv.org/abs/2406.14598.

31. Xiong, T., et al., 2024, 'LLaVA-Critic: Learning to Evaluate Multimodal Models'. Available at: https://arxiv.org/abs/2410.02712.

32. Yue, X., et al., 2023, 'Automatic Evaluation of Attribution by Large Language Models'. Available at: https://aclanthology.org/2023.findings-emnlp.307/.

33. Lee, S., et al., 2024, 'Aligning Large Language Models by On-Policy Self-Judgment'. Available at: https://arxiv.org/abs/2402.11253.

34. Liu, M., et al., 2024, 'X-Eval: Generalizable Multi-aspect Text Evaluation via Augmented Instruction Tuning with Auxiliary Evaluation Aspects'. Available at: https://arxiv.org/abs/2311.08788.

35. Ke, P., et al., 2024, 'CritiqueLLM: Towards an informative critique generation model for evaluation of large language model generation'. Available at: https://arxiv.org/abs/2311.18702.

36. Hu, X., et al., 2024, 'Themis: A Reference-free NLG Evaluation Language Model with Flexibility and Interpretability'. Available at: https://arxiv.org/abs/2406.18365.

37. Li, Z., et al., 2024, 'Split and Merge: Aligning Position Biases in LLM-based Evaluators'. Available at: https://arxiv.org/abs/2310.01432.

38. Liu, Y., et al., 2023, 'G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment'. Available at: https://arxiv.org/abs/2303.16634.