LLM Comparator: Visual Analytics for Side-by-Side Evaluation of Large Language Models

Minsuk Kahng
kahng@google.com
Google Research
Atlanta, GA, USA

Ian Tenney
iftenney@google.com
Google Research
Seattle, WA, USA

Mahima Pushkarna
mahimap@google.com
Google Research
Cambridge, MA, USA

Michael Xieyang Liu
lxieyang@google.com
Google Research
Pittsburgh, PA, USA

James Wexler
jwexler@google.com
Google Research
Cambridge, MA, USA

Emily Reif
ereif@google.com
Google Research
Seattle, WA, USA

Krystal Kallarackal
kallarackal@google.com
Google Research
Cambridge, MA, USA

Minsuk Chang
minsukchang@google.com
Google Research
Seattle, WA, USA

Michael Terry
michaelterry@google.com
Google Research
Cambridge, MA, USA

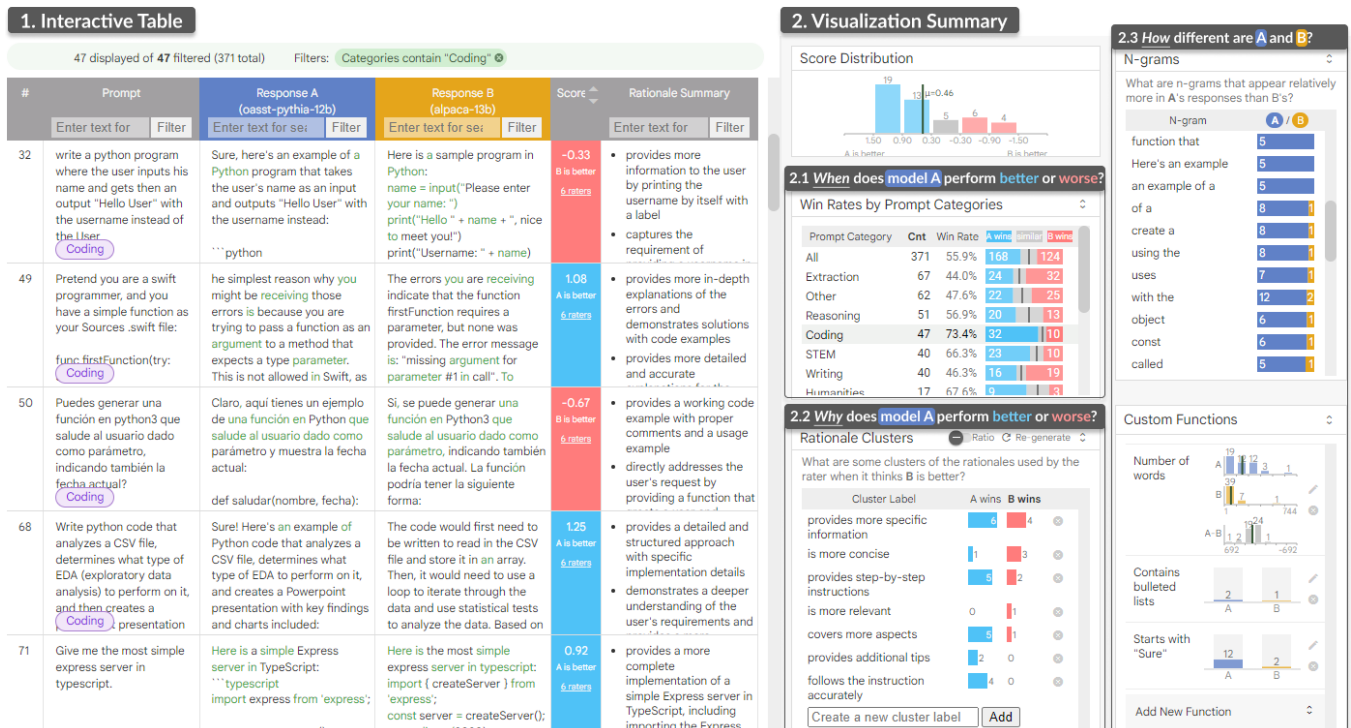Lucas Dixon
ldixon@google.com
Google Research
Paris, France

Figure 1: *LLM Comparator* enables model developers and researchers to interactively analyze results from *automatic side-by-side evaluation* of large language models (LLMs). To evaluate the quality of responses from an LLM (A), users can compare them with those from a baseline LLM (B). The tool's *interactive table (1)* enables users to inspect individual prompts and their responses in details, and its *visualization summary (2)* supports analytical workflows that help understand *when (2-1)* and *why (2-2)* a model performs better or worse and *how (2-3)* the two models' responses are different.

ABSTRACT

Automatic side-by-side evaluation has emerged as a promising approach to evaluating the quality of responses from large language models (LLMs). However, analyzing the results from this evaluation approach raises scalability and interpretability challenges. In this paper, we present *LLM Comparator*, a novel visual analytics tool for interactively analyzing results from automatic side-by-side evaluation. The tool supports interactive workflows for users to understand when and why a model performs better or worse than a baseline model, and how the responses from two models are qualitatively different. We iteratively designed and developed the tool by closely working with researchers and engineers at Google. This paper details the user challenges we identified, the design and development of the tool, and an observational study with participants who regularly evaluate their models.[1]

CCS CONCEPTS

• **Human-centered computing → Visualization systems and tools**; **Interactive systems and tools**.

KEYWORDS

Visual analytics, generative AI, large language models, machine learning evaluation, side-by-side evaluation

1 INTRODUCTION

Large language models (LLMs) are constantly being trained and tuned by researchers and engineers for performance improvements. This involves adjusting model parameters, adding or removing training data examples, and other changes to the training procedure. A critical challenge here is evaluating whether an updated model performs sufficiently well to supplant a baseline model.

However, the evaluation of LLMs poses unique challenges. Unlike traditional machine learning models, which are evaluated against ground-truth answers, it is impractical to establish ground truth responses for LLMs due to the fact that they tend to generate long freeform text. Therefore, a widely-used approach to evaluate LLMs is to ask humans to rate the output from the model by comparing with that from a baseline model. While effective, it does not scale to many experiments, as it is expensive to obtain human ratings. To mitigate these challenges, *automatic side-by-side evaluation (a.k.a., AutoSxS, LLM-as-a-judge)* has emerged as a promising approach to evaluating LLMs [7, 20]. This approach involves asking another LLM to compare the quality of the outputs of two models. The prompt typically asks the LLM to select which response is better in terms of their quality. Additionally, the LLM might be asked to justify its selection.

To deeply understand practitioners' workflows that utilize automatic side-by-side evaluation, we have had conversations with researchers and engineers in a variety of teams at Google. We learned that while aggregated scores from these LLM-based automatic raters provide a quick initial assessment of model performance as a single number, people often express strong needs for

further analysis of the rater results. In particular, they often face challenges with interpretability and sensemaking. For example, they want to understand *why* a particular model received a win rate of 56%. Additionally, they need to deduce the types of scenarios in which a model will perform well or poorly.

In this paper, we present *LLM Comparator*, a novel interactive tool for researchers and engineers to analyze automatic side-by-side evaluation results in a scalable manner. It provides interactive, visual analytics workflows that enable users to obtain a visual summary of the results from the side-by-side ratings, while simultaneously delve into corresponding individual examples to explore the qualitative behavioral differences between the two models. Specifically, the tool visualizes slice-level performances (**when** the model performs better), rationale summaries (**why** it is better), and n-grams and custom functions (**how** they are different).

LLM Comparator has been successfully integrated into evaluation pipelines for large teams at Google. Since our initial announcement to select teams, the tool has attracted over 400 users within its first three months, facilitating the analysis of over 1,000 unique automatic side-by-side experiments. Feedback from these users has enabled us to iteratively improve the tool. Section 3 describes our latest prototype, and Section 4 presents a qualitative user study that evaluates it with six participants who regularly use automatic raters for their model developments.

2 CURRENT WORKFLOWS & DESIGN GOALS

In this section, we discuss the current practice of LLM evaluations and our design goals for building a new interactive tool for analyzing the automatic side-by-side evaluations of LLMs. We base our discussion on our informal conversations with over 20 people from multiple teams at Google.

2.1 Current Practice of LLM Evaluations

We discovered that automatic side-by-side evaluation (i.e., AutoSxS [7], LLM-as-a-judge [20]) is one of the most prevalent evaluation practices. Once model developers train a new model, they would like to quickly evaluate it by running AutoSxS libraries, before conducting more costly and time-consuming human evaluations. Specifically, the process consists of:

- **Baseline models:** A baseline model to compare is set as a currently-deployed version of the model or one that has been shown to perform well (e.g., PaLM 2 [3]).
- **Prompt sets:** People select one of the available prompts sets, each typically ranging from hundreds to thousands. Each prompt is commonly tagged with categories (e.g., coding).
- **Individual ratings:** The AutoSxS libraries take an input prompt, a response from the model to test, and a response from the baseline. It returns a Likert-scale rating (e.g., "A is much better," "B is slightly better," etc.) accompanied by a brief rationale. The Likert-scale ratings are transformed into numeric scores (e.g., "A is much better" equals 1.5, "A is better" equals 1.0, "B is much better" equals -1.5). To counteract the stochastic nature of the scores, the process is often repeated multiple times, and the final score is calculated as the average of these repetitions [12, 20].

#	Prompt	Response A (oasst-pythia-12b)	Response B (alpaca-13b)	Score	Rationale Summary

360	Write me a python code of insertion sort algorithm	Here's a Python implementation of the insertion sort algorithm:
		
		def insertionSort(array):
		  for i in range(1, len(array)):
		    key = array[i]
		    j = i - 1
			def insertionSort(arr):
			  n = len(arr)
			
			  for i in range(1, n):
			    min_index = i
			    for j in range(0, i):
			      if arr[min_index] > arr[j]:
				0.33 A is better 6 raters	• provides more explanations of how the code works
					• correctly implements the logic of the insertion sort
					• includes more detailed comments explaining the purpose and functioning of the algorithm

Individual Rater Scores

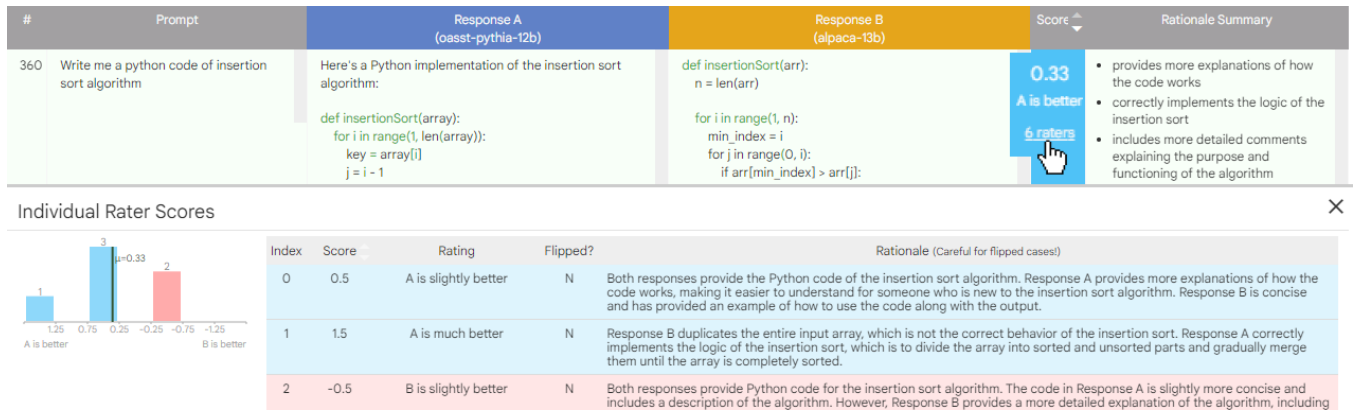| Index	Score	Rating	Flipped?	Rationale (Careful for flipped cases!)
0	0.5	A is slightly better	N	Both responses provide the Python code of the insertion sort algorithm. Response A provides more explanations of how the code works, making it easier to understand for someone who is new to the insertion sort algorithm. Response B is concise and has provided an example of how to use the code along with the output.
1	1.5	A is much better	N	Response B duplicates the entire input array, which is not the correct behavior of the insertion sort. Response A correctly implements the logic of the insertion sort, which is to divide the array into sorted and unsorted parts and gradually merge them until the array is completely sorted.
2	-0.5	B is slightly better	N	Both responses provide Python code for the insertion sort algorithm. The code in Response A is slightly more concise and includes a description of the algorithm. However, Response B provides a more detailed explanation of the algorithm, including

Figure 2: Users can inspect the individual ratings to see the detailed rationales used by the raters.

- **Aggregated metrics:** The libraries calculate metrics from the ratings across a set of prompts, with average scores and win rates being the most prevalent. A win rate can be defined as the proportion of scaled rating scores that are above or below the threshold (A wins if score > 0.3; B wins if < -0.3; tie otherwise).

2.2 User Challenges in Evaluation Analysis

We have identified common workflows of how model developers and researchers analyze the results from these AutoSxS libraries:

- No specialized tools existed for the analysis of AutoSxS results. Typically, results are loaded into spreadsheets—with one row for each input prompt, and the columns are prompt, response A, response B, and the score. Some practitioners load the data into *computational notebooks* (e.g., Jupyter, Colab).
- Practitioners eyeball individual examples (i.e., prompts and LLM responses) to interpret evaluation results and compare differences between responses from two models qualitatively. They either randomly select examples to examine or sort them by scores using spreadsheets and examine those with exceptionally high or low scores. It is often challenging to read them, as spreadsheets are not designed for long texts.
- They have strong interests in computing metrics (e.g., average scores, win rates) by slices (e.g., prompt categories) to identify which slices underperform or outperform relative to others. Then they often want to inspect examples in these slices, but it requires switching between different tools.
- For further analysis, practitioners compute additional features from response texts (e.g., number of tokens) and aggregate the feature values across the examples by using computational notebooks or other tools.

Importantly, both detailed examination of individual examples and analysis of aggregated data are essential; however existing tools fail to connect these two types of analyses.

2.3 Design Goals

Given the current practice of running LLM evaluations and analyzing their results outlined above, we distill the following design goals for building tools for side-by-side evaluation analysis:

DG1. Facilitate interactions between the aggregated information and individual examples. This will enable users to identify their slices of interests in diverse ways and examine specific prompts and their response pairs in details.

DG2. Provide workflows to uncover answers to the following analytical questions:

2-1. When: In which situations does model A perform better than model B?

2-2. Why: What are the common rationales used by the raters? Why does it say one model is better than another?

2-3. How: How are the responses between two models different? What qualitative patterns can be observed? Can these patterns be used to inform data or model improvements?

DG3. Perform the analysis of evaluation results at scale for a large number of prompts. This will allow users to more confidently discern the performance differences between two models.

3 VISUALIZATION DESIGN & DEVELOPMENT

In this section, we introduce *LLM Comparator*, a interactive tool for the side-by-side comparison of LLMs. Figure 1 shows our tool's interface for a scenario where a researcher who develops a new LLM evaluates its performance by comparing it to a baseline model.[2] For a given prompt set, they obtain side-by-side ratings using another LLM[3] that compares the quality of response pairs. The tool consists of two main panels: the (1) an *interactive table* for detailed individual example inspection and (2) a *visualization summary* for overviews and filtering options that support the users' analytical workflows.

---

[2]This figure uses data of prompts and responses from two selected models from LMSys Chatbot Arena Conversation dataset [20].
[3]We used Google Cloud's Generative AI APIs on Vertex AI available at https://cloud.google.com/vertex-ai/docs/generative-ai/learn/overview.

3.1 Interactive Table

Each row in the table represents a prompt, its corresponding responses from two models, the rater's score, and a rationale summary. Below we highlight a few unique features of the interactive table:

- **Overlapping word highlights.** To facilitate quick and easy comparison of two response texts, it highlights overlapping words between the two as green text (e.g., "def insertionSort" in Figure 2).
- **Rationale summary.** The rationales are often too lengthy to read in full, particularly with multiple raters involved (see Figure 2, bottom). To address this challenge, we employ another LLM to summarize them into a bulleted list (in Figure 2, rightmost column). If one receives six ratings and the average outcome favors A (with 4 for A being better and 2 for B), we ask the LLM to summarize the four cases favoring A.
- **Individual rating details.** The average score is displayed on the table row, with an option to view the details of individual ratings if desired (i.e., by clicking "6 raters" link as shown in Figure 2).
- **Color coding scheme.** We represent A with indigo and B with orange. Also, to represent the rater's decisions, we use blue to indicate rows where the rater prefers A, red where the rater prefers B, and gray to denote ties.

3.2 Visualization Summary

The visualization summary panel features several components designed to support the analytical workflows of users:

Score Distribution. Upon first encountering the summary metric (e.g., average score = 0.46), users would often ask about its distribution. To help answer this question, we display a simple histogram for the distribution of scores (ranging from 1.5 to -1.5).

Win Rates by Prompt Category (*when*). To answer the common analysis question of in what scenarios model A outperforms or underperforms compared to model B (DG2-1), we present a visualization of performance across different prompt categories. This facilitates users to identify categories with notably higher or lower scores, informing which data examples to inspect further. In Figure 1 (**2-1** on the right), a high-scoring category "Coding" is selected.

Rationale Clusters (*why*). To help users understand the rationales behind the rater's decisions (DG2-2), we condense a large array of rationales into several representative themes. While there are various methods to produce these themes, for example, by running clustering on all rationale bullets and subsequently labeling the clusters, we opted for a novel LLM-based approach that performs better and runs faster based on our testing. Specifically, we first ask a different LLM to generate a set of diverse and representative cluster labels given a sample of the rationale bullets, inspired by recent work [18, 21]. We then assign each rationale bullet to clusters (represented by their labels) based on embedding similarity,[4] i.e., if the cosine similarity between the bullet and the label exceeds an empirically determined threshold, it is considered a match. Note that each bullet can be assigned to multiple clusters.

---

[4]To assign rationale bullets into clusters shown in Figure 1, we used Google Cloud's text-embeddings APIs at https://cloud.google.com/vertex-ai/docs/generative-ai/embeddings/get-text-embeddings.
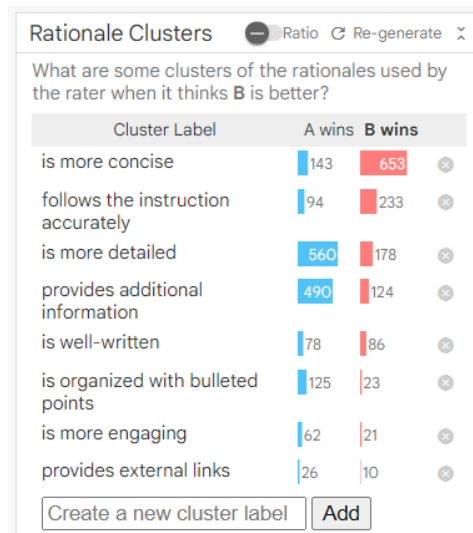

Figure 3: The rationale clusters view presents a list of rationales that are frequently used by the automatic rater. Users can dynamically add ones to compare the occurrences of relevant rationales between the two models.

As shown in Figure 3, for each cluster label, the tool counts the number of instances where model A is determined to be better, and vice versa. By sorting these counts, users will be able to identify common rationales used by the rater. Moreover, it can also be particularly useful to examine the ratio between the count for A and B. For instance, if a certain rationale cluster (e.g., "is more concise") shows a significantly higher count for B, users can hypothesize that B's responses are generally more concise than A.

Users can interact with the clustering results by dynamically adding or removing individual clusters, or regenerating the entire cluster set for the filtered examples. In addition, when combined with the prompt category filter, users can inspect the different rationales used by the rater for different types of prompts.

N-grams and Custom Functions (*how*). To grasp the nuances of the rationales, users must be able to examine individual instances. For instance, if the automatic rater states that "B is more organized," a user might still wonder about what it means to be "organized." While users can directly inspect individual response pairs from the table, we provide two additional techniques:

- **N-gram Analysis.** The tool presents frequently occurring n-grams (n=1 to 7) in responses from either A or B, compared to its counterpart (e.g., "Here's an example" appears 150 times in A's responses while appearing only 3 times in B's).
- **Custom Functions.** Users can either define regular expressions (e.g., newline character followed by dash or star indicating bulleted items) or JavaScript function expressions (e.g., word count specified by "output.split(/\s+/).length"). Upon specifying these expressions, they immediately apply to each individual response and return either boolean or numeric values. For boolean values (e.g., whether it contains bulleted items), the tool visualizes the results as percentage
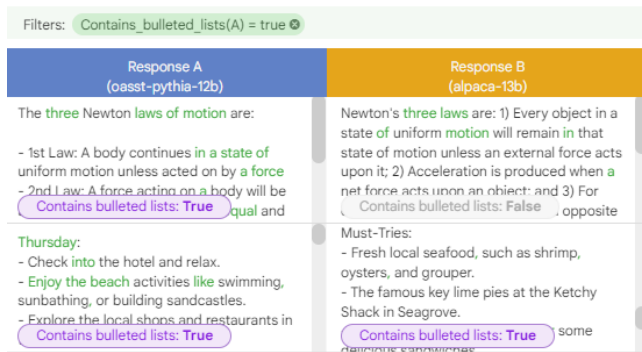
Figure 4: Users can dynamically create functions that apply to responses. A function specified as a regular expression (i.e., `"\n([*-])\s"`) checks if each response contains bulleted lists, and their results are displayed as purple chips.

bar charts; for numeric values (e.g., word count), it displays histograms. They can be displayed on top of the responses when selected (as shown in Figure 4).

3.3 Implementation

LLM Comparator is implemented as a web-based application. Its preprocessing module loads a data file that stores the results from the AutoSxS libraries containing a list of prompts with response pairs and the ratings with rationales. Then it calls an LLM to summarize rationales into bullet points, generate cluster labels, and compute embeddings to be used for cluster assignments. The server, written in Python, loads this preprocessed data file and then transmits it to the client in JSON format. Once data is loaded into the client, all computations, such as filtering, sorting, and cluster assignments, are performed dynamically on web browser. The client-side code is written in TypeScript using the Lit framework.[5] When a user requests to regenerate rationale clusters, the server invokes calls to an LLM using a RPC call.

3.4 System Deployment

LLM Comparator has been developed based on iterative feedback from many engineers, researchers, and data scientists at Google. Since our initial announcement to select internal LLM development teams, the tool has attracted over 400 users, facilitating the analysis of over 1,000 distinct side-by-side evaluation experiments. In addition, it has been deployed on evaluation pipelines for large teams who develop LLMs for their products. The final-stage of the pipelines performs preprocessing for *LLM Comparator*. When the pipeline is complete, users see a direct link to our tool on the platform interface.

While the earlier versions of the tool featured the interactive table and a subset of the visualization summary components described in this section, the latest prototype updated based on the user feedback offers the full functionalities described in this section, including the rationale clusters and N-grams analysis. In the next section, we present our evaluation of this latest prototype.

4 OBSERVATIONAL STUDY

We conducted an observational study to investigate how users would use *LLM Comparator* presented in Section 3.

4.1 Study Setup

Participants. We recruited six participants (P1-6) from our company, consisting of software engineers, researchers, and data scientists who are directly involved with teams dedicated to developing LLM-based products. All participants had experience conducting automatic side-by-side evaluations within the past month. Additionally, some had previous experience with earlier iterations of our tool, e.g., versions without the rationale clusters feature.

Study Protocol. Each study session was conducted remotely over video conferencing and took around 45 minutes. After participants sign the consent form, we first conducted a 10-minute interview focused on the participant's role in model evaluation. This was followed by a 5 to 10-minute tutorial on the tool's features. Participants were then asked to use the tool while think aloud to analyze a recent side-by-side evaluation run they had performed on internal evaluation platforms. The session concluded with a short reflective interview, and the participants were compensated with $25 USD. We analyzed the results through thematic analysis.

4.2 Key Usage Patterns

Our study revealed the following interesting usage patterns.[6]

4.2.1 ***Example-first deep dive.*** P1 and P2 invested significant time in reading prompts and responses to gain insights from the results, especially when they first launched the tool. Driven by the overall metric favoring Model B (baseline model), P2 wanted to inspect low-scoring examples for Model A (their model). They sorted the examples by the rater score and scanned the rows one by one. They meticulously read prompts to find one they can familiarize with, and then read and compared response pairs. P2 said this process is crucial because the automatic rater is not always right, so they need to make sure if the rater is working correctly. P1 used an interesting alternative strategy. They concealed the score column and predicted the automatic rater's scores, mimicking the process for human raters.

After spending time analyzing examples, participants began formulating hypotheses about behavioral differences between the two models. P2 noticed a case where Model A's response succinctly only include the code for a prompt about coding, while B additionally provided detailed explanations. This difference caught P2's attention, because it might be caused by a specific change they have made to their model. To further find similar cases, they filtered examples by prompt category (i.e., coding) and quickly found several other examples that exhibit similar patterns. Moreover, the rationale clusters view reveals one named "Provide clear explanations" with much higher counts for B, further confirming their hypothesis.

4.2.2 ***Prior experience-based testing.*** P3, P4, and P5 leveraged their prior knowledge to identify undesirable model behaviors. P3 sought to find responses containing phrases like "I'm sorry" or

"unfortunately" which often signal a model's refusal to answer the tasks specified in prompts. They need to discern whether these are genuinely unanswerable prompts for LLMs or areas for potential improvement to deliver accurate responses. Similarly, P5 noted the desire to detect cases where LLMs generate unnecessary phrases (e.g., starting sentences with "here is", overusing bold text) to internally optimize their objectives, which is a known behavior of LLMs [2].

Participants reported maintaining a collection of such undesirable patterns for testing purposes (similar to performing *testing* in software engineering [19]), and used the tool to determine if these patterns were present in either side of the models. Specifically, they used the tool's N-grams and custom function features to make initial searches for these phrases. Subsequently, they used visualizations to compare the occurrences of these phrases across the two models. For example, after finding a noticeable difference between counts, they meticulously examined the corresponding individual responses and leveraged the rationale summaries and clusters to check whether the automatic raters paid attention to this information.

4.2.3 **Rationale-centric top-down exploration.** The rationale clusters view enabled a diverse set of ways to analyze data that were previously unavailable. P2 had used the earlier versions of the tool before, and they used it primarily for inspecting individual examples. While the only way to understand the rater's rationales was selecting an example first and opening the detailed rating results view, the updated version introduces rationale clusters, providing new methods for in-depth data investigation to validate their hypotheses about the model's behavior. In addition, P3, who had also used the tool many times before, first searched for specific keywords like "sorry" as described earlier. However, they later noticed one of the rationale clusters "Avoids harmful content", and by applying a filter for this cluster, they were pleased to see several interesting keywords from the N-grams components. These keywords include those which they had to manually search for individually, including "I'm sorry."

Participants also adopted a more exploratory approach actively engaging with the visualizations to discover interesting patterns. Coordinated views that are dynamically updated capture their attention and spark curiosity. For instance, P6 noticed a category with a significantly higher win rate from the chart. Applying a filter for this category, they could naturally form new hypotheses from one of the rationale clusters about conciseness. This led them to use a custom function for word count and identified responses that are very short and problematic.

4.3 Discussions and Future Opportunities

In addition to the above usage patterns, we uncovered opportunities to further enhance users' analysis workflows.

LLM-based analysis for high-level differences. While the N-grams and custom functions are effective for analyzing qualitative differences, people have additional needs to assess high-level attributes (e.g., safety, verbosity). To address this limitation, we may employ yet another LLM, similar to prior work [11]. However, this approach brings substantial complexity due to the extensive LLM calls, particularly for the dynamic evaluation of large prompt

sets. Exploring practical solutions to mitigate this bottleneck would greatly enhance the feasibility and scalability of LLM-based evaluations for further analysis.

Pre-configured undesirable patterns. Participants expressed a strong desire for the tool to be pre-configured with specific unwanted patterns, to avoid manually defining new functions or searching for individual keywords. For example, P3 particularly cared about identifying the issue of repeated sentences in LLM outputs, highlighting the importance to be able to easily detect and flag such occurrences.

Improving rationale clustering. The pipeline for clustering rationales relies on LLM calls, which could be error-prone. Also, it could be less than ideal to use embedding similarity for clustering assignments, as embeddings reflect not only *semantic* but also *syntactic* similarity. Alternative computational approaches and more advanced interactions (in addition to what we implemented, e.g., adding new clusters) would boost the robustness and efficiency of this pipeline.

5 RELATED WORK

Visual Analytics for Machine Learning Interpretability. In the past decade, a variety of methodologies and tools for machine learning analysis have been developed from the visualization community. These include early works that emphasized the importance of visualizing individual data points [1] and supporting slice-level analysis [10, 13], tools that utilized various interpretability methods to explain individual predictions [16], and methods and techniques for model comparison [5, 9, 14, 17]. As LLMs have emerge, tools targeting specific types of language models have been introduced [6, 8, 14, 15].

Interactive Tools for LLM Evaluations. With ChatGPT's rise in 2023, interactive tools for LLM evaluations and comparisons have begun to appear in late 2023. A recent preprint, ChainForge [4], presented a flexible framework to perform comparisons with user-specified functions. Another recent work, EvalLM [11], presented a tool for interactively performing LLM-based evaluations by user-defined criteria. Different from these concurrently developed approaches, our work focuses on the visual analysis and interpretation of large-scale evaluations for industry practitioners.

6 CONCLUSION

We presented a new interactive tool for analyzing results from side-by-side LLM evaluation methods. The tool aimed at enabling users to analyze when and why a model performs better or worse than a baseline model and how they behave differently. Our observational study indicated that the tool enables participants to form hypotheses about the automatic ratings, verify known model behaviors, and analyze qualitative differences between model responses.

ACKNOWLEDGMENTS

REFERENCES

[1] Saleema Amershi, Max Chickering, Steven M Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. 2015. ModelTracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI)*. 337–346. https://doi.org/10.1145/2702123.2702509

[2] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565* (2016). https://arxiv.org/abs/1606.06565

[3] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. PaLM 2 technical report. *arXiv preprint arXiv:2305.10403* (2023). https://arxiv.org/abs/2305.10403

[4] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena Glassman. 2023. ChainForge: A Visual Toolkit for Prompt Engineering and LLM Hypothesis Testing. *arXiv preprint arXiv:2309.09128* (2023). https://arxiv.org/abs/2309.09128

[5] Angie Boggust, Brandon Carter, and Arvind Satyanarayan. 2022. Embedding Comparator: Visualizing differences in global structure and local neighborhoods via small multiples. In *27th International Conference on Intelligent User Interfaces (IUI)*. 746–766. https://doi.org/10.1145/3490099.3511122

[6] Richard Brath, Daniel Keim, Johannes Knittel, Shimei Pan, Pia Sommerauer, and Hendrik Strobelt. 2023. The Role of Interactive Visualization in Explaining (Large) NLP Models: from Data to Inference. *arXiv preprint arXiv:2301.04528* (2023). https://arxiv.org/abs/2301.04528

[7] Google Cloud. 2024. Perform automatic side-by-side evaluation. https://cloud.google.com/vertex-ai/docs/generative-ai/models/side-by-side-eval

[8] Adam Coscia and Alex Endert. 2023. KnowledgeVIS: Interpreting Language Models by Comparing Fill-in-the-Blank Prompts. *IEEE Transactions on Visualization and Computer Graphics* (2023).

[9] Michael Gleicher, Aditya Barve, Xinyi Yu, and Florian Heimerl. 2020. Boxer: Interactive comparison of classifier results. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 181–193. https://arxiv.org/abs/2004.07964

[10] Minsuk Kahng, Pierre Y Andrews, Aditya Kalro, and Duen Horng Chau. 2017. ActiVis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2017), 88–97. https://doi.org/10.1109/TVCG.2017.2744718

[11] Tae Soo Kim, Yoonjoo Lee, Jamin Shin, Young-Ho Kim, and Juho Kim. 2023. EvalLM: Interactive Evaluation of Large Language Model Prompts on User-Defined Criteria. *arXiv preprint arXiv:2309.13633* (2023). https://arxiv.org/abs/2309.13633

[12] Tianqi Liu, Zhen Qin, Junru Wu, Jiaming Shen, Misha Khalman, Rishabh Joshi, Yao Zhao, Mohammad Saleh, Simon Baumgartner, Jialu Liu, et al. 2024. LiPO: Listwise Preference Optimization through Learning-to-Rank. *arXiv preprint arXiv:2402.01878* (2024). https://arxiv.org/abs/2402.01878

[13] Yao Ming, Huamin Qu, and Enrico Bertini. 2018. RuleMatrix: Visualizing and understanding classifiers with rules. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 342–352. https://doi.org/10.1109/TVCG.2018.2864812

[14] Hendrik Strobelt, Benjamin Hoover, Arvind Satyanarayan, and Sebastian Gehrmann. 2021. LMdiff: A visual diff tool to compare language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*. https://arxiv.org/abs/2111.01582

[15] Hendrik Strobelt, Albert Webson, Victor Sanh, Benjamin Hoover, Johanna Beyer, Hanspeter Pfister, and Alexander M Rush. 2022. Interactive and visual prompt engineering for ad-hoc task adaptation with large language models. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2022), 1146–1156. https://arxiv.org/abs/2208.07852

[16] Ian Tenney, James Wexler, Jasmijn Bastings, Tolga Bolukbasi, Andy Coenen, Sebastian Gehrmann, Ellen Jiang, Mahima Pushkarna, Carey Radebaugh, Emily Reif, and Ann Yuan. 2020. The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*. https://arxiv.org/abs/2008.05122

[17] Junpeng Wang, Liang Wang, Yan Zheng, Chin-Chia Michael Yeh, Shubham Jain, and Wei Zhang. 2022. Learning-from-disagreement: A model comparison and visual analytics framework. *IEEE Transactions on Visualization and Computer Graphics* (2022). https://arxiv.org/abs/2201.07849

[18] Zihan Wang, Jingbo Shang, and Ruiqi Zhong. 2023. Goal-Driven Explainable Clustering via Language Descriptions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 10626–10649. https://doi.org/10.18653/v1/2023.emnlp-main.657

[19] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* 48, 1 (2020), 1–36. https://arxiv.org/abs/1906.10742

[20] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Neural Information Processing Systems (NeurIPS): Datasets and Benchmarks Track*. https://arxiv.org/abs/2306.05685

[21] Ruiqi Zhong, Charlie Snell, Dan Klein, and Jacob Steinhardt. 2022. Describing differences between text distributions with natural language. In *International Conference on Machine Learning (ICML)*. PMLR, 27099–27116. https://proceedings.mlr.press/v162/zhong22a.html