
Cross-Model Assessment: How LLMs Judge Each Other's Responses via RAG

Anthony Gasbarro

Department of Electrical and Computer Engineering
University of Hawaii at Manoa
Honolulu, HI 96822
agasbarr@hawaii.edu

Chris Aguilar

Department of Electrical and Computer Engineering
University of Hawaii at Manoa
Honolulu, HI 96822

Maxwell Pauly

Department of Electrical and Computer Engineering
University of Hawaii at Manoa
Honolulu, HI 96822

Abstract

Retrieval-Augmented Generation (RAG) systems leverage external knowledge bases to significantly enhance the ability of Large Language Models (LLMs) to generate accurate and contextually relevant information. This study investigates the performance evaluation of Retrieval-Augmented Generation (RAG) systems using multiple Large Language Models (LLMs) as judges. RAG systems enhance the ability of LLMs to generate accurate and contextually relevant information by leveraging external knowledge bases. Our analysis, based on 149 questions, highlights significant discrepancies in evaluation scores provided by different LLM judges for the same generated responses. These differences often stem from hallucinations, parsing errors, or biases unique to individual models. Importantly, these inconsistencies demonstrate that biases and idiosyncrasies inherent to each LLM significantly influence evaluation outcomes, rendering single-model assessments unreliable. Our research corroborates findings of LLM text evaluation biases that have been published in the past.

1 Introduction

Large language models (LLMs) have exploded into the mainstream with many competitors in the space, each with their own quirks and bespoke training methods. The largest competing models such as GPT, Claude, Hugging Face, and LLaMA are being deployed across a broad range of applications such as, customer support, creative writing, and data summarization. However, evaluating the quality, accuracy, and consistency of an LLM's responses remains an open challenge. Scalable methods to make a fine-grained assessment via automated metric-based scoring often falls short of capturing the nuance associated with answers spanning multiple knowledge domains or abbreviating information from an external data source. A promising direction is offered through the usage of other LLMs to evaluate the outputs of a tested LLM. This offers scalability and nuanced assessment but introduces

new questions regarding the reliability of biases each of these LLM-based judges. Specifically, we must understand how different LLM evaluators compare when scoring the same generated response.

2 Background

Retrieval Augmented Generation (RAG) systems are a useful tool that combine the strengths of information retrieval and large language models. By retrieving relevant information from external knowledge bases, such as databases or documents, RAG systems enhance the accuracy, factuality, and context-awareness of LLM-generated outputs. This significantly impacts industries like customer service, where chatbots can provide more accurate and personalized responses. In academia, RAG systems are invaluable for research, enabling researchers to efficiently synthesize information from vast amounts of scientific literature, accelerating the pace of discovery. By bridging the gap between LLMs and real-world information, RAG systems are poised to enhance various fields and serve as an additional tool for LLM systems. A modular "cookbook" from HuggingFace serves as a building block for exploration into RAG architecture, shown in Figure 1.

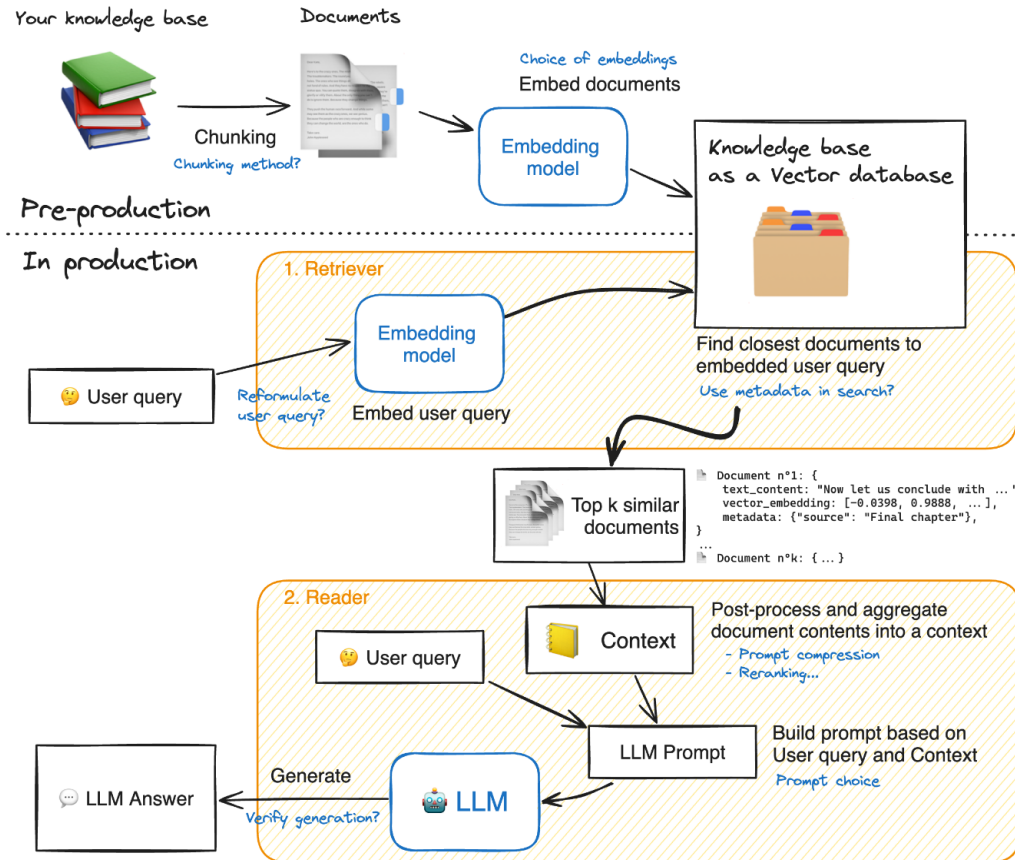


Figure 1: Flowchart of general RAG pipeline (2)

2.1 Evaluating RAG Systems

To develop and explore the effectiveness of any system, a way to evaluate its performance is necessary. Evaluation typically focuses on two main dimensions: retrieval performance and generation quality, each of which can significantly influence the final output. Retrieval performance is often assessed using information retrieval metrics such as precision at K (P@K), recall at K (R@K), mean reciprocal rank (MRR), and normalized discounted cumulative gain (nDCG). These metrics gauge how well the retriever identifies relevant documents from the knowledge base. Generation quality, on the other hand, is measured using text evaluation metrics such as BLEU, ROUGE, METEOR, and newer

embedding-based methods like BERTScore or BLANC to evaluate fluency, relevance, and fidelity compared to reference answers. Some evaluations also incorporate human judgment for a qualitative analysis of factual accuracy, coherence, and informativeness.

2.2 LLMs as Evaluators

Large Language Models (LLMs) have shown to be effective as automatic evaluators with simple prompting and in-context learning (10). They have become a popular choice for Natural Language Processing (NLP) evaluation tasks, including the evaluation of RAG systems. If you take a foundation model, a reference Question/Answer (QA) pair, and the output of a RAG system tasked with answering the question from the QA pair, then the foundation model can 'score' the quality of the output with regards to the reference QA pair.

2.3 The Impact of Foundation Model Choice on RAG System Evaluation

The effects of using different evaluator models is the main topic of interest in this paper. Here we will motivate why a stakeholder would care about these effects, and a summary of the knowledge already produced by the RAG Evaluation industry.

The Cognitive Bias Benchmark for LLMs as Evaluators (COBBLER) (11) was recently developed to measure the biases in LLM evaluation outputs. An example of an observed bias is the tendency for models to evaluate their own outputs with higher ratings compared to another model's outputs, often without regard to the actual quality of the response. This showcases the need to use a different foundation model as the evaluator. The COBBLER study found that LLMs are biased text quality evaluators, exhibiting strong indications on their bias benchmark. They also found that human and machine preferences are misaligned, according to their Rank-Biased Overlap (RBO) score of 44 percent.

A study on the general LLM evaluation in NLP (12) also supports this claim, showing LLMs are biased evaluators and prefer text with lower perplexity, exhibiting a familiarity bias. The study also shows LLM evaluators experience the "anchor effect", a phenomenon where a previous evaluation score or piece of information acts as a reference point, influencing subsequent evaluations and leading to inconsistent results.

Introducing biases could influence the evaluation of RAG systems, so knowing about their existence and quantifying their effects is paramount to the RAG industry. We accept that the evaluator models have bias and include multiple evaluators in an attempt to normalize the response. We analyze the accuracy level, and the difference between the two evaluators which allows us to conclude whether the RAG system is working properly, or if the evaluator bias is skewing the evaluation results.

2.4 Benchmark Selection

We conducted a literature review in order to find an applicable benchmark. The pre-made HF cookbook was set to use the "Faithfulness" (13) metric, used to evaluate the factual consistency of the generated answer against the given context. It is calculated from the answer and retrieved context. The generated answer is regarded as faithful if all the claims made in the answer can be inferred from the given context. This is calculated by identifying the set of claims from the generated answer. The claims are then cross checked with the given context to determine if it can be inferred from the context. See the faithfulness score equation below.

$$\text{Faithfulness score} = \frac{\text{\# of inferrable generated claims}}{\text{total \# of claims}}$$

This benchmark was chosen as a method of comparison between different evaluators, due to the prompting being standardized between model evaluations. A number of other benchmarks were considered including context precision, context recall, context entities recall, noise sensitivity, response relevancy, faithfulness, multimodal faithfulness, and multimodal relevance (13). We leave analysis of these benchmarks for future work.

We also note the COBBLER benchmark (11) was considered, which ultimately tests for the RAG system’s output accuracy, which was used in the paper to make bias conclusions. This benchmark was not used because it ultimately did not align with the project’s goal.

2.5 Methodology

The methodology begins by loading the source document, which is then used by selected generation model to generate Question-Answer (QA) pairs. After generating the pairs, the generation model filters them based on predefined criteria like accuracy and relevance, producing a refined list. The source document is then embedded for efficient document retrieval. A selected reader model processes a given question and generates an answer based on the embedded document. The answer from the reader model, along with the original answer from generation model, is evaluated for accuracy by an LLM evaluator. The evaluation results are graphed to compare the performance of different evaluators, allowing an analysis of how various models judge the same QA pair and revealing any discrepancies between each evaluation that may inform whether there may be issues with answering the question.

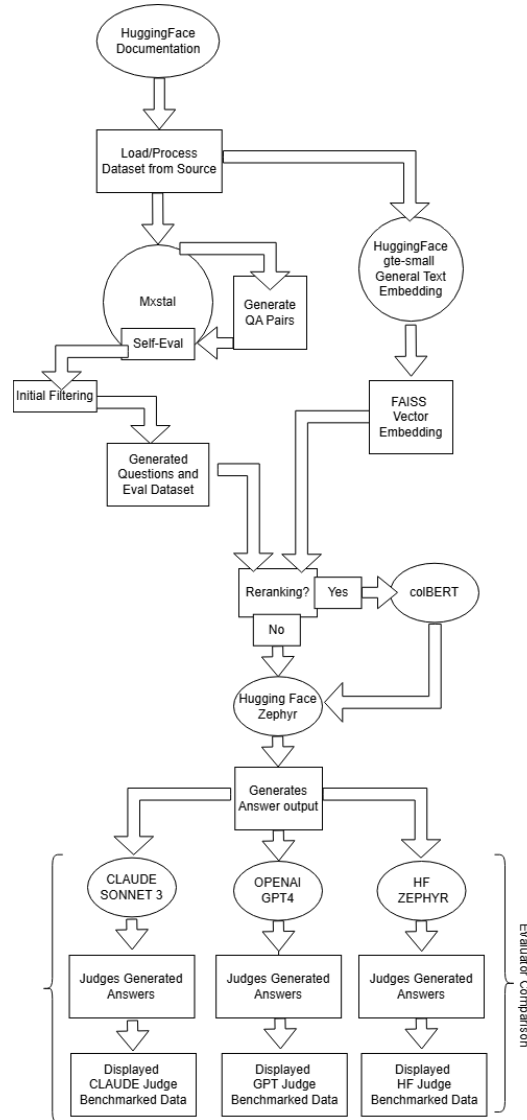


Figure 2: Flowchart of RAG Evaluation Pipeline

2.5.1 Source Documentation

The source document used in this project is the Hugging Face Documentation Dataset, which is a dataset that provides comprehensive technical documentation on various machine learning models, tools, and tasks within the Hugging Face ecosystem. We chose this dataset because it offers a rich, structured collection of information that is relevant to natural language processing and machine learning, making it ideal for generating and evaluating Question-Answer (QA) pairs in our RAG framework. The detailed and diverse content of the documentation allows us to test the system's ability to generate accurate answers to a wide range of queries, from basic model descriptions to more complex technical details.

2.5.2 Question-Answer Pair Generation

We use the source documentation to generate Question-Answer (QA) pairs with Mixtral-9x7B-Instruct-v0.1, a model that is available to use, which is highly suited for creating factoid questions that require factually accurate, clear, and concise answers. The core model is a high quality sparse mixture-of-experts model that outperforms GPT3.5 and Llama 2 70B on most benchmarks, such as MMLU, HellaSwag, and ARC Challenge. It's instruction-tuned model, Mixtral-9x7B-Instruct-v0.1 is a fine tuned version, specifically made for tasks that require high precision in language generation, such as question answering. The generation process is actuated using a structured prompt, which instructs the model to formulate a factoid question based on a given context, ensuring that the question is specific, concise, and answerable by a single piece of factual information from the context. The model is also guided to create questions that align with how users typically ask questions in search engines, avoiding phrases like "according to the passage" or "context."

2.5.3 Question-Answer Pair Filtration

The filtration process begins after the initial question-answer pairs are generated and critiqued for quality. Each QA pair undergoes a series of evaluations based on three main criteria: groundedness, relevance, and standalone clarity. These evaluations are provided by the same model that generates them, Mixtral-9x7B-Instruct-v0.1, which rates the QA pair on a scale from 1 to 5 for each criterion. Groundedness assesses how well the question can be answered unambiguously using the provided context, relevance, in this specific context, evaluates how useful the question is for machine learning developers working within the Hugging Face ecosystem, and standalone clarity measures how context-independent the question is, ensuring that it makes sense on its own without additional context. After the critiques are generated, the QA pairs are filtered based on the scores: any pair that receives a score lower than 4 in any of the evaluation categories is discarded. This ensures that only high-quality QA pairs—those that are clear, relevant, and grounded in the source material—are retained for further analysis.

2.5.4 Source Document Embedding

The source document embedding process involves converting the raw text from the source documents into a format suitable for efficient retrieval. This is done by splitting the documents into smaller, manageable chunks to ensure that the retrieval system can effectively match relevant pieces of information to user queries. First, the documents are processed and split based on a specific chunk size, ensuring that each chunk contains a coherent piece of text. Once the documents are divided, they are embedded using a pre-trained embedding model, thenlper/gte-small, which is part of the Hugging Face embeddings library and is designed to generate semantic representations of text and converts the textual content into high-dimensional vectors. These embeddings make it easier to search for similar information in response to a query. The embeddings are then indexed using FAISS, a popular library for efficient similarity search, which organizes the embeddings into a searchable structure. The indexed embeddings allow for rapid retrieval of relevant information, enabling the RAG system to quickly access the most pertinent parts of the document when generating responses.

2.5.5 Reader Answer

The Reader LLM is responsible for generating a concise and relevant answer to a user's query by leveraging the information contained within the context provided. In this process, the relevant documents are first retrieved from the knowledge base, which is indexed using document embeddings.

The Reader LLM then processes these documents and the user’s question, using a structured prompt to guide the model in providing a focused response. The prompt directs the model to only answer the question asked, ensuring that the response is both specific and directly tied to the retrieved content. If the question cannot be answered with the given context, the model is instructed not to generate a response. In this case, the Reader LLM used is the HuggingFace Zephyr-7B model. This model is fine-tuned for tasks like text generation and question answering, allowing it to generate high-quality, fact-based responses. Notably, Zephyr-7B has achieved good performance on chat benchmarks for 7B parameter models, requiring no human annotation. In particular, results on MT-Bench show that Zephyr-7B surpasses Llama2-Chat-70B, one of the best open-access RLHF-based model.(14) The Reader LLM ensures that the generated answers are grounded in the content of the retrieved documents, and any irrelevant or ambiguous information is excluded from the final response.

2.5.6 Evaluator Rating

After a question is answered by the Reader LLM, evaluators such as Claude, GPT-4, or Zephyr-7B are tasked with comparing the generated answer to the reference answer from the dataset. The evaluation is based on a standardized rubric, which is structured to assess the accuracy, relevance, and factual correctness of the response. The rubric uses a score from 1 to 5, with a score of 1 representing an incorrect or irrelevant answer, and a score of 5 indicating that the answer is completely accurate and factual. The rubric provides the following breakdown:

- Score 1: Completely incorrect or inaccurate.
- Score 2: Mostly incorrect or lacking factual accuracy.
- Score 3: Somewhat correct, with minor inaccuracies.
- Score 4: Mostly correct with minor flaws or omissions.
- Score 5: Fully correct and factual, aligning perfectly with the reference answer.

This grading system allows evaluators to provide fine-grained feedback on each answer, helping identify whether the generated response is correct, relevant, or needs improvement. The reason for choosing this rubric is to ensure consistency and reliability in evaluating answers, particularly when working with complex RAG systems where the quality of answers can vary based on the retrieved documents. By using this clear and structured rubric, we can compare how different evaluators rate the same answers, helping to mitigate potential biases and discrepancies in their assessment. The evaluation process involves comparing the generated answers with the true answers provided in the dataset, and each evaluator assesses the answers independently. This process is repeated for all answers in the dataset, and feedback is generated for each response. By leveraging multiple evaluators, we ensure a comprehensive and nuanced understanding of the model’s performance across different evaluation criteria

2.5.7 Results Comparison

Finally, the accuracy of the evaluator’s score is calculated by normalizing the raw score provided by the evaluator, which ranges from 1 to 5. To convert this into a more interpretable scale, the score is first adjusted by subtracting 1, so the minimum score becomes 0 and the maximum score becomes 4. This adjusted score is then divided by 4, scaling it to a range between 0 and 1. Finally, the normalized score is multiplied by 100 to convert it into a percentage, which represents the accuracy of the generated answer. The scores are then aggregated into an average accuracy for that configuration.

$$\text{Accuracy}_i = \frac{\text{Evaluator Score}_i - 1}{4} \times 100$$

$$\text{Overall Accuracy} = \frac{1}{n} \times \sum_{i=1}^n \text{Accuracy}_i$$

Where n is the number of questions evaluated, and i is the index of the question being evaluated. After processing all the evaluations, the results are grouped into different configurations (e.g., reader models, embeddings, chunk sizes) and the average accuracy is calculated for each configuration. The average accuracy scores are then plotted on a bar chart, where the x-axis represents different RAG

configurations (such as the type of reader model, chunk size, or embeddings used), and the y-axis shows the accuracy as a percentage. This graph allows for a visual comparison of how different configurations perform, highlighting the most accurate setups for the given task.

2.6 Testing and Evaluation

The current iteration begins with a test dataset of Hugging Face Documentation Dataset then uses Mixtral’s *Mixtral-8x7B-Instruct-v0.1* model to generate question and answer pairs about the dataset. A critique agent is then instructed to grade the question/answer pairs generated and low scoring entries are flagged for removal. Langchain is then used to split the dataset documents of the knowledge-base into chunks then FAISS indexes the data for retrieval. The embeddings are created using Hugging Face’s General Text Embeddings (GTE) *gte-small*. The Reader LLM in this case is HuggingFace’s *zephyr-7b-beta* which is responsible for answering the questions generated by *Mixtral-8x7B-Instruct-v0.1*. The final judging is performed by OpenAI’s *gpt-4-1106-preview* model and is given an instruction prompt on how to evaluate a response then it grades the HuggingFace *zephyr-7b-beta*’s responses for accuracy.

In our recent testing phase, we evaluated the performance of our RAG model by generating 500 question-and-answer (Q/A) pairs based on the preexisting huggingface doc dataset provided by HuggingFace. Following the generation process, these pairs underwent a quality filtering stage to remove any subpar entries leaving 149 high-quality Q/A pairs for further analysis.

The retained pairs were processed through the RAG pipeline, as illustrated in Figure 2. This pipeline facilitated a comprehensive assessment of the model’s performance by enabling us to fine-tune various parameters, including chunk size, reranking strategies, query evaluation metrics, the embedding model used for document retrieval, the number of retrieved documents, and the thresholds for filtering low-quality responses.

Models acting as judges were given the an evaluation prompt for instruction on how to judge a given answer relative to the reference answer shown in Figure 3

```

1  EVALUATION_PROMPT = """###Task Description:
2  An instruction (might include an Input inside it), a response to evaluate, a reference answer that gets a score of 5,
   ↳ and a score rubric representing a evaluation criteria are given.
3  1. Write a detailed feedback that assess the quality of the response strictly based on the given score rubric, not
   ↳ evaluating in general.
4  2. After writing a feedback, write a score that is an integer between 1 and 5. You should refer to the score rubric.
5  3. The output format should look as follows: \"Feedback: {{write a feedback for criteria}} [RESULT] {{an integer
   ↳ number between 1 and 5}}\"
6  4. Please do not generate any other opening, closing, and explanations. Be sure to include [RESULT] in your output.
7
8  ###The instruction to evaluate:
9  {instruction}
10
11  ###Response to evaluate:
12  {response}
13
14  ###Reference Answer (Score 5):
15  {reference_answer}
16
17  ###Score Rubrics:
18  [Is the response correct, accurate, and factual based on the reference answer?]
19  Score 1: The response is completely incorrect, inaccurate, and/or not factual.
20  Score 2: The response is mostly incorrect, inaccurate, and/or not factual.
21  Score 3: The response is somewhat correct, accurate, and/or factual.
22  Score 4: The response is mostly correct, accurate, and factual.
23  Score 5: The response is completely correct, accurate, and factual.
24
25  ###Feedback: ""
26

```

Figure 3: Instruction prompt given to evaluation LLMs for critiquing answers

We conducted two sets of tests: the first evaluated the generated responses in their original form, while the second employed a reranking process using the *colBERT* model to assess the relevance and accuracy of these responses before they were passed to our "Reader" model, *zephyr-7b-beta*, for final answer generation. This approach allowed us to determine the effectiveness of reranking in enhancing the quality of responses, providing valuable insights into the model’s performance under different configurations.

Following this, we employed OpenAI’s *gpt-4-1106-preview* model to evaluate the answers. GPT-4 analyzed each response for accuracy and relevance in relation to the original questions and the embedded knowledge from our dataset. Using tailored evaluation prompts, GPT-4 provided scores and feedback as an overall accuracy.

3 Results

The results of *claude-3-sonnet*, *gpt-4-1106-preview*, and *zephyr-7b-beta*’s evaluations for each question were averaged and plotted via an interpretation function, the generated data in *Appendix A: Figures 7, 8, and 9*. This data and the full code for testing is available on the project’s GitHub Repository

The results in both the *claude-3-sonnet* and *zephyr-7b-beta* performed slightly better when using a reranker by 0.4% and 2.3% respectively but performed slightly worse by about 0.2% in the case of *gpt-4-1106-preview*. We chose to compare the reranker scores for each model as it is considered standard practice to implement re-ranking during RAG and the comparison can be seen in Figure 4.

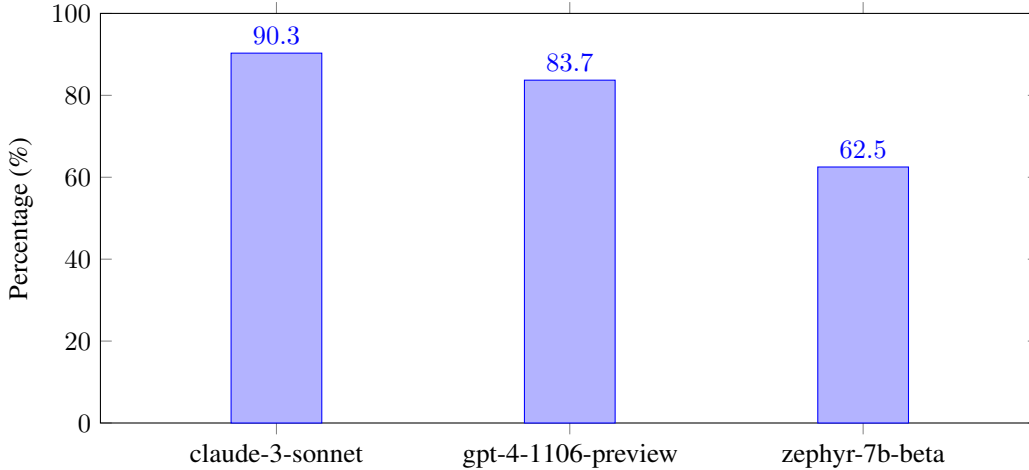


Figure 4: Average Score in percent as judged by each model on a rubric scaled 1 to 5

The feedback associated with the score that each model provides gives insight into the decision making of each score given and enables a qualitative analysis of how each model differs in its assessment. Figure 5a shows an example of a question that all judges gave a perfect score and gave similar feedback while Figure 5b shows an example of a question where the answer given was considered poor and lacking proper context by all models but *claude-3-sonnet* and *zephyr-7b-beta* scored the answer a 3 while *gpt-4-1106-preview* scored it at a 1.

A qualitative analysis of Figure 5b shows that all models feedback were correct to assess that the answer does not provide a correct class name and also does not provide enough information or commit to a specific answer. But in this case, *gpt-4-1106-preview* hallucinates an exact name of *StableDiffusionImageVariationPipeline* as the correct output function in the text but it is not actually specified as the correct answer to the given question *What is the name of the class that is the output of the ImagePipeline?*. It is assumed that this hallucination of false correct answer results in the much lower score that the model evaluates for this instance.

Results of the testing showed that on average, *gpt-4-1106-preview* evaluated answers with lower scores than *claude-3-sonnet* by about 7% but found that *zephyr-7b-beta* graded itself the most harshly scoring about 24.5% lower than the average of the other two judges. *zephyr-7b-beta*’s large decrease in scores could possibly be a result of other limitations of the testing configuration and is addressed in the following section.


```

1 "eval_score_Claude-3-Sonnet": "5",
2 "eval_feedback_Claude-3-Sonnet": "<Feedback> The response accurately explains that bounding boxes should be in (x0,
  ↳ y0, x1, y1) format, with (x0, y0) being the upper left corner and (x1, y1) being the lower right corner. It also
  ↳ correctly mentions that the bounding boxes need to be normalized to a 0-1000 scale before passing them to the
  ↳ model or processor, matching the details in the reference answer. The response covers all the key points
  ↳ required.",
3 "eval_score_gpt-4-1106-preview": "5",
4 "eval_feedback_gpt-4-1106-preview": "Feedback: The response correctly identifies the format for bounding boxes in
  ↳ pyesseract as (x0, y0, x1, y1) and accurately describes the meaning of these coordinates. It also correctly notes
  ↳ the need for normalization of the bounding boxes on a 0-1000 scale, which is in line with the reference answer.
  ↳ Additionally, the response provides extra relevant information about the option to use one's own OCR engine and
  ↳ the associated setting of `apply_ocr` to `False`. This additional information does not detract from the accuracy
  ↳ of the response with respect to the question asked. Therefore, the response meets the criteria for a score of 5 as
  ↳ it is completely correct, accurate, and factual based on the reference answer.",
5 "eval_score_HuggingFaceH4/zephyr-7b-beta": "5",
6 "eval_feedback_HuggingFaceH4/zephyr-7b-beta": "Feedback: The response accurately describes how bounding boxes should
  ↳ be formatted in pyesseract and highlights the importance of normalizing them before passing to LayoutLMModel or
  ↳ LayoutLMv2Processor."

```

(a) Example of all evaluation of judges in agreement

```

1 "eval_score_Claude-3-Sonnet": "3",
2 "eval_feedback_Claude-3-Sonnet": "I do not have enough information in the given context to determine the specific
  ↳ class that is the output of the ImagePipeline. The context mentions several different ImagePipeline classes like
  ↳ ImageClassificationPipeline, ImageSegmentationPipeline, etc., but does not clearly specify which one the question
  ↳ is referring to or what the output class should be. Without more context about the particular ImagePipeline being
  ↳ referred to, I cannot provide a definitive answer.",
3 "eval_score_gpt-4-1106-preview": "1",
4 "eval_feedback_gpt-4-1106-preview": "The response does not provide the correct class name as per the reference answer.
  ↳ The response indicates uncertainty and does not commit to a specific class name, instead discussing the various
  ↳ ImagePipeline classes mentioned in the context without concluding with the exact class name
  ↳ 'pipelines.ImagePipelineOutput'. The response fails to identify the correct class name from the context provided,
  ↳ which was explicitly mentioned in Document 0 as 'StableDiffusionImageVariationPipeline'. Therefore, the response
  ↳ does not meet the criteria for correctness, accuracy, or factuality based on the reference answer.",
5 "eval_score_HuggingFaceH4/zephyr-7b-beta": "3\n\nFeedback: The question does not provide enough information to
  ↳ determine",
6 "eval_feedback_HuggingFaceH4/zephyr-7b-beta": "Based on the provided context, the question asks for the name of the
  ↳ class that is the output of the ImagePipeline. However, no specific ImagePipeline is mentioned, which makes it
  ↳ unclear which class is intended. To answer the question, we would need more context or clarification about which
  ↳ ImagePipeline is being referred to. Without this information, it is impossible to provide a definitive answer."

```

(b) Example of dissenting opinion where one judges grades lower than other models

Figure 5: Comparison of model evaluations with both agreement and dissent

3.1 Limitations

The LLM RAG Evaluation was heavily based on HuggingFace’s Open-Source AI Cookbook. This particular configuration communicates with each model via API calls with a given instruction prompt. Although the prompt specifies an exact format for the LLM to respond with, inconsistencies still occur frequently due to variations in the LLM’s responses and possible special characters and delimiters or when API communication errors occur. The results gathered were parsed into *JSON* files to allow for simplified accumulation and computation of evaluation results but differences in API standards for each required bespoke adjustments in certain cases. The largest inconsistencies in results were found when attempting to communicate with HuggingFace’s own *zephyr-7b-beta* model. An example of an inconsistent response and parsing error can be seen in Figure 6.

```

1 "eval_score_Claude-3-Sonnet": "5\n\nThe assistant's response is completely correct, accurate, and factual based on the
  ↳ reference answer. The assistant provided the exact steps needed to specify the dataset and configuration to process
  ↳ with Apache Beam, including setting the `DATASET_NAME` and `CONFIG_NAME` environment variables. The response
  ↳ accurately reflects the information given in the context.",
2 "eval_feedback_Claude-3-Sonnet": "",
3 "eval_score_gpt-4-1106-preview": "5",
4 "eval_feedback_gpt-4-1106-preview": "Feedback: The response correctly identifies the need to set `DATASET_NAME` and
  ↳ `CONFIG_NAME` environment variables to specify the dataset and configuration for Apache Beam processing, which
  ↳ aligns with the reference answer. It also correctly includes additional steps for setting up the Google Cloud
  ↳ Platform information and Python requirements, which, while not explicitly mentioned in the reference answer, are
  ↳ relevant and factual based on the provided context. The response goes beyond the reference answer by providing
  ↳ extra, relevant information without deviating from the accuracy of the answer. Therefore, the response meets the
  ↳ criteria for a score of 5.",
5 "eval_score_HuggingFaceH4/zephyr-7b-beta": "0",
6 "eval_feedback_HuggingFaceH4/zephyr-7b-beta": "[In order to specify the dataset and configuration for processing with
  ↳ Apache Beam, you need to set the `DATASET_NAME` and `CONFIG_NAME` environment variables. For example,
  ↳ `DATASET_NAME=wikimedia` and `CONFIG_NAME=20220301.en`. If you want to avoid specifying the configuration name in
  ↳ `load_dataset()`, you can set a default configuration name using the `DEFAULT_CONFIG_NAME` variable."
7

```

Figure 6: Example of inconsistent response and issues parsing outputs

4 Conclusion

RAG systems offer an efficient way to significantly enhance the capability of LLMs by integrating external knowledge bases for generating accurate information. In this study, we assessed the performance of a RAG model through evaluations conducted by various LLM-based judges, aiming to understand the reliability and biases of each model as an evaluator. Our analysis revealed instances of substantial discrepancies in the evaluation scores provided by different models for the same generated responses. These discrepancies were often attributed to hallucinations by the evaluation models or errors arising from API responses and parsing issues. However, these errors and hallucinations affected each model differently, introducing significant variability that did not average out over the course of our 149-question evaluation.

We conclude that the inherent biases and idiosyncrasies of individual LLMs and their APIs can have a substantial impact on evaluation outcomes, making it impractical to rely on any single model for assessing a RAG model’s responses. Our research corroborates findings of LLM text evaluation biases that have been published in the past (12). Instead, employing a voting-based evaluation system that aggregates judgments from multiple models is essential to mitigate outliers and ensure a more balanced and reliable assessment may be explored in the future.

References

- [1] Mistral.ai. (2023). Mixtral: Mixtral of Experts. Retrieved from <https://mistral.ai/news/mixtral-of-experts/>.
- [2] Hugging Face. (2023). Hugging Face Learning: Cookbook. Retrieved from <https://huggingface.co/learn/cookbook/en/index>.
- [3] Hugging Face. (2023). Hugging Face Dataset Repository: Huggingface_doc Dataset by M-RIC. Retrieved from https://huggingface.co/datasets/m-ric/huggingface_doc.
- [4] Facebook Research. (2023). FAISS: A Library for Efficient Similarity Search and Clustering of Dense Vectors. Retrieved from <https://github.com/facebookresearch/faiss>.
- [5] LangChain. (2023). LangChain Documentation: Introduction. Retrieved from <https://python.langchain.com/docs/introduction/>.
- [6] Hugging Face. (2023). GTE-Small by Thenlper. Retrieved from <https://huggingface.co/thenlper/gte-small>.
- [7] Wang, A., Diao, S., Wang, Y., Huang, Y., Zhou, S., Feng, Y., & Ma, X. (2023). InstructE: Teaching Language Models to Personalize to You. *arXiv preprint arXiv:2308.03281*. Retrieved from <https://arxiv.org/abs/2308.03281>.
- [8] Feng, Y., Liu, Z., Wang, Y., Xiao, L., Huang, Y., Ma, X., & He, J. (2023). RAGQL: Reinforcement Learning with Augmented Feedback from Questions and Language Models. *arXiv preprint arXiv:2306.05685*. Retrieved from <https://arxiv.org/abs/2306.05685>.
- [9] OpenAI. (2023). OpenAI Platform: Model Documentation. Retrieved from <https://platform.openai.com/docs/models>.
- [10] Style Over Substance: Evaluation Biases for Large Language Models. Retrieved from <https://arxiv.org/abs/2307.03025>.
- [11] Benchmarking Cognitive Biases in Large Language Models as Evaluators. Retrieved from <https://arxiv.org/pdf/2309.17012>.
- [12] Large Language Models are Inconsistent and Biased Evaluators. Retrieved from <https://arxiv.org/pdf/2405.01724>.
- [13] Ragas RAG Evaluation Metrics. Retrieved from https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/.
- [14] Hugging Face. (2023). Zephyr: Direct Distillation of LM Alignment. Retrieved from <https://arxiv.org/abs/2310.16944>.

A Figures and data

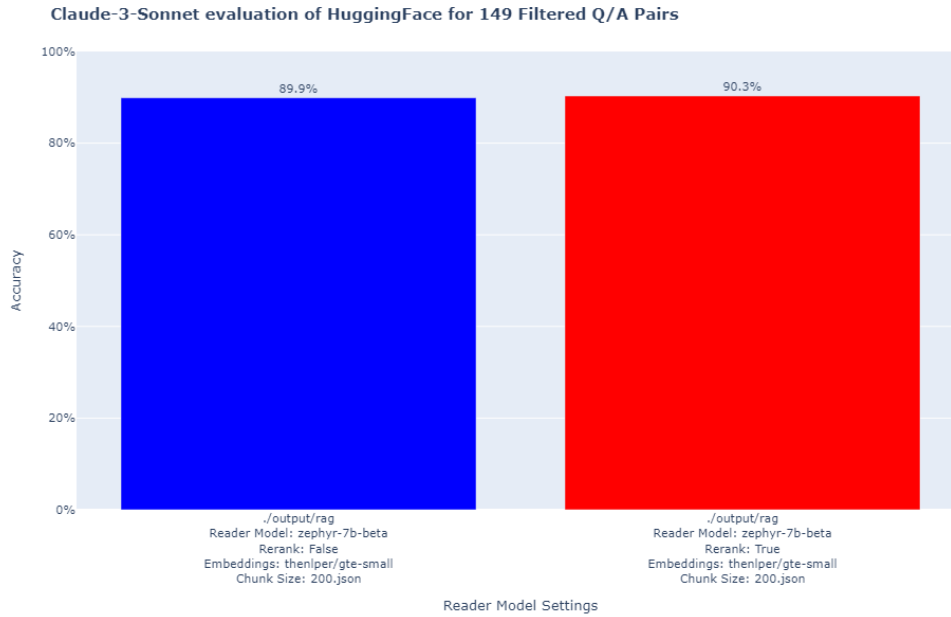


Figure 7: Claude-3-Sonnet evaluation results for HuggingFace-zephyr-7b-beta

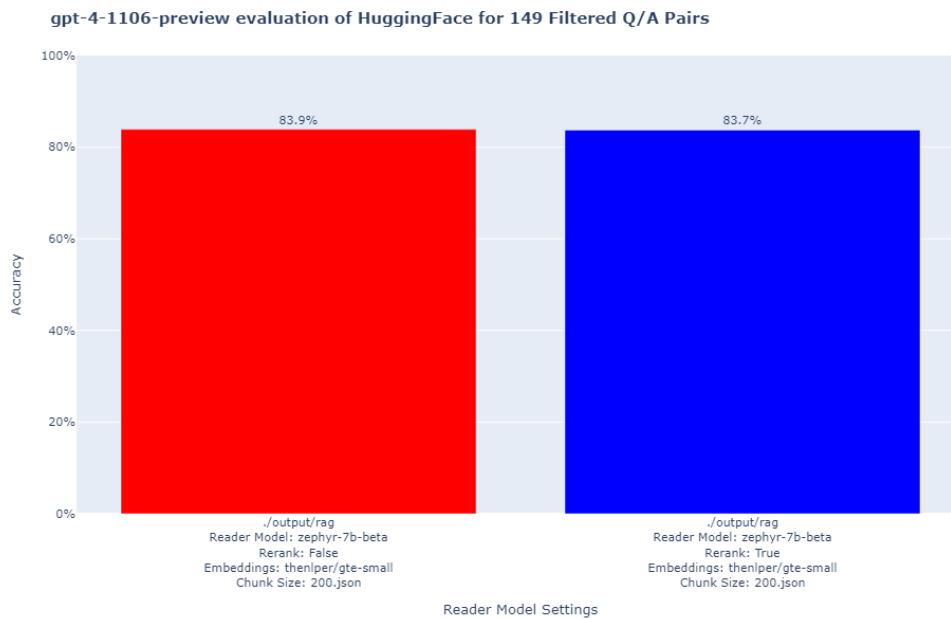


Figure 8: ChatGPT4-1106-preview evaluation results for HuggingFace-zephyr-7b-beta

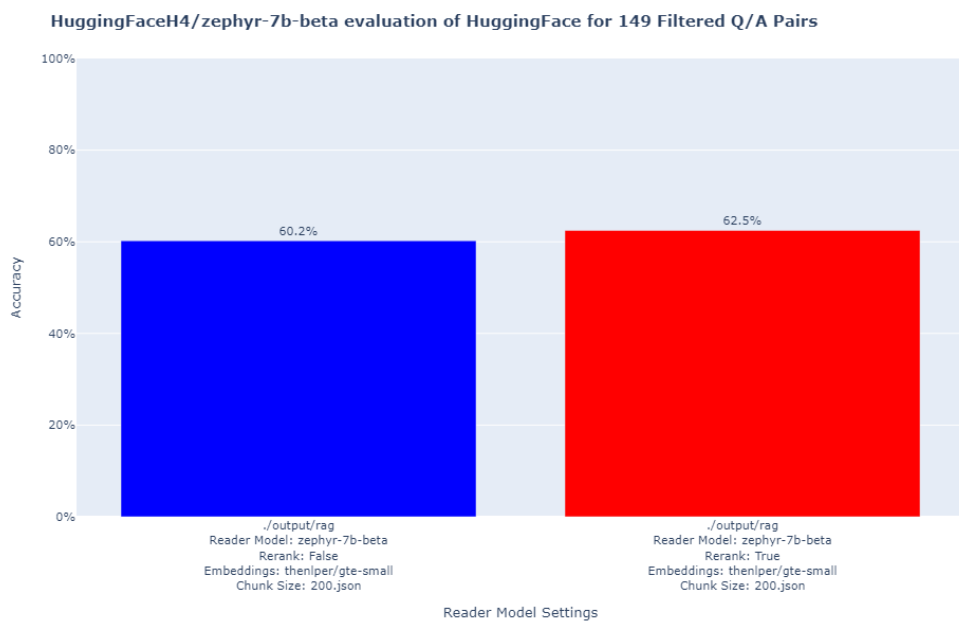


Figure 9: HuggingFace-zephyr-7b-beta results when evaluating its own answers

```

1 {
2     "question": "What is the name of the class that is the output of the ImagePipeline?",
3     "true_answer": "pipelines.ImagePipelineOutput",
4     "source_doc": "huggingface/diffusers/blob/main/docs/source/en/api/pipelines/dit.md",
5     "generated_answer": "\n<system>\nUsing the information contained in the context,\ngive a comprehensive answer to
    the question.\nRespond only to the question asked, response should be concise and relevant to the
    question.\nProvide the number of the source document when relevant.\nIf the answer cannot be deduced from the
    context, do not give an answer.</s>\n<user>\nContext:\n\nExtracted documents:\nDocument 0::\nimage =
    output.images[0]\nimage.save(\"./shiba-inu.jpg\")\n\n# debug\n\n# `pipeline.decoder_pipe` is a regular
    StableDiffusionImageVariationPipeline instance.\n# It is used to convert clip image embedding to latents, then
    fed into VAE decoder.\nprint(pipeline.decoder_pipe.__class__)\n# <class 'diffusers.pipelines.stable_diffusion`
    .pipeline_stable_diffusion_image_variation.StableDiffusionImageVariationPipeline'>\nDocument 1::\n\n# Community
    pipelines\n\nCommunity pipelines are any ['DiffusionPipeline'] class that are different from the original
    implementation as specified in their paper (for example, the ['StableDiffusionControlNetPipeline'] corresponds
    to the [Text-to-Image Generation with ControlNet Conditioning](https://arxiv.org/abs/2302.05543) paper). They
    provide additional functionality or extend the original implementation of a pipeline.\nDocument 2::\n\nThe
    'AutoPipeline' class is designed to simplify the variety of pipelines in Diffusers. It is a generic,
    *task-first* pipeline that lets you focus on the task. The 'AutoPipeline' automatically detects the correct
    pipeline class to use, which makes it easier to load a checkpoint for a task without knowing the specific
    pipeline class name.\n\n<Tip>\n\nTake a look at the [AutoPipeline](../api/pipelines/auto_pipeline) reference
    to see which tasks are supported. Currently, it supports text-to-image, image-to-image, and
    inpainting.\n\n<Tip>\n\nThis tutorial shows you how to use an 'AutoPipeline' to automatically infer the
    pipeline class to load for a specific task, given the pretrained weights.\nDocument 3::\n\npipe =
    AutoPipelineForText2Image.from_pretrained(\"kandinsky-community/kandinsky-2-2-decoder\", torch_dtype=torch.fl
    oat16)\npipe.prior.load_attn_procs(output_dir)\npipe.enable_model_cpu_offload()\n\nnprompt='A robot
    pokemon, 4k photo'\nimage =
    pipe(prompt=prompt).images[0]\nimage.save(\"robot_pokemon.png\")\nimage\n\n```\n\n### Training with
    xFormers:\nDocument 4::\n\n### ImageClassificationPipeline\n\n[[autodoc]] ImageClassificationPipeline\n
    - __call__\n
    - all\n\n### ImageSegmentationPipeline\n\n[[autodoc]] ImageSegmentationPipeline\n
    - __call__\n
    - all\n\n### ImageToImagePipeline\n\n[[autodoc]] ImageToImagePipeline\n
    - __call__\n
    - all\n\n### ObjectDetectionPipeline\n\n[[autodoc]] ObjectDetectionPipeline\n
    - __call__\n
    - all\n\n### VideoClassificationPipeline\n\n[[autodoc]] VideoClassificationPipeline\n
    - __call__\n
    - all\n\n### ZeroShotImageClassificationPipeline\n\nDocument 5::\n\n```\npython\nfrom transformers import pipeline\nfrom
    optimum.intel.openvino import OVModelForImageClassification\n\nnov_model =
    OVModelForImageClassification.from_pretrained(save_dir)\nnov_pipe = pipeline(\"image-classification\",
    model=nov_model, image_processor=processor)\n\noutputs =
    ov_pipe(\"http://farm2.staticflickr.com/1375/1394861946_171ea43524_z.jpg\")\n\nprint(outputs)\n\n```\n\nDocument
    6::\n\npipe_prior = DiffusionPipeline.from_pretrained(output_dir, torch_dtype=torch.float16)\n\nprior_components
    = {\"prior_\" + k: v for k,v in pipe_prior.components.items()}\n\npipe =
    AutoPipelineForText2Image.from_pretrained(\"kandinsky-community/kandinsky-2-2-decoder\", **prior_components,
    torch_dtype=torch.float16)\n\npipe.enable_model_cpu_offload()\n\nprompt='A robot pokemon, 4k photo'\nimages =
    pipe(prompt=prompt, negative_prompt=negative_prompt).images\n\nimages[0]\n\n```\n\n---\n\nNow here is the question you
    need to answer.\n\nQuestion: What is the name of the class that is the output of the
    ImagePipeline?\n\n</s>\n\n<assistant>\n\nThe question asks for the name of the class that is the output of the
    ImagePipeline. Based on the provided context, there are multiple ImagePipeline classes mentioned, including
    ImageClassificationPipeline, ImageSegmentationPipeline, ImageToImagePipeline, ObjectDetectionPipeline, and
    VideoClassificationPipeline. However, none of these classes are specifically referred to as the output of an
    ImagePipeline. Therefore, it is unclear which ImagePipeline class is being referred to in this question.
    Without further context, it is impossible to provide a definitive answer. If the question is referring to a
    specific ImagePipeline class, please provide more context to clarify which one is being referred to. If the
    question is asking for the output class of any ImagePipeline, please specify that as well.\"

```

Figure 10: Question and Answer for dissenting opinion example