Optimizing and Evaluating Enterprise Retrieval-Augmented Generation (RAG): A Content Design Perspective

Sarah Packowski
spackows@ca.ibm.com
IBM
Canada

Inge Halilovic
ingeh@us.ibm.com
IBM
United States

Jenifer Schlotfeldt
jschlot@us.ibm.com
IBM
United States

Trish Smith
smith@ca.ibm.com
IBM
Canada

ABSTRACT

Retrieval-augmented generation (RAG) is a popular technique for using large language models (LLMs) to build customer-support, question-answering solutions. In this paper, we share our team's practical experience building and maintaining enterprise-scale RAG solutions that answer users' questions about our software based on product documentation. Our experience has not always matched the most common patterns in the RAG literature. This paper focuses on solution strategies that are modular and model-agnostic. For example, our experience over the past few years - using different search methods and LLMs, and many knowledge base collections - has been that simple changes to the way we create knowledge base content can have a huge impact on our RAG solutions' success. In this paper, we also discuss how we monitor and evaluate results. Common RAG benchmark evaluation techniques have not been useful for evaluating responses to novel user questions, so we have found a flexible, "human in the lead" approach is required.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; **Natural language generation**; • **Applied computing** → *Document management and text processing*.

KEYWORDS

Retrieval-augmented generation, RAG, Large language models

ACM Reference Format:
Sarah Packowski, Inge Halilovic, Jenifer Schlotfeldt, and Trish Smith. 2024. Optimizing and Evaluating Enterprise Retrieval-Augmented Generation (RAG): A Content Design Perspective. In *Proceedings of 8th International Conference on Advances in Artificial Intelligence (ICAAI '24).* ACM, New York, NY, USA, 6 pages.

ICAAI '24, October 2024, London, UK

ACM ISBN 979-8-4007-1801-4/24/10

1 INTRODUCTION

Retrieval-augmented generation (RAG) is an effective way to use large language models (LLMs) to answer questions while avoiding hallucinations and factual inaccuracy[12, 20, 46]. Basic RAG is simple: 1) search a knowledge base for relevant content; 2) compose a prompt grounded in the retrieved content; and 3) prompt an LLM to generate output. For the retrieval step, one approach dominates the literature: 1) segment content text into chunks; 2) index vectorized chunks for search in a vector database; and 3) when generating answers, ground prompts in a subset of retrieved chunks[13]. Our RAG solutions don't always use vector databases for search.

Wikipedia has long been influenced by and had an influence on scientific research [21, 41]. With respect to RAG, Wikipedia is a dominant source of knowledge base content for training data and benchmarks, including: 2WikiMultiHopQA, AmbigQA, ASQA, DART, FEVER, HotpotQA, KILT, MuSiQue, Natural Questions, NoMIRACL, PopQA, SQuAD, StrategyQA, SuperGLUE, TriviaQA, WikiAsp, WikiBio, WikiEval, and Wizard of Wikipedia[8, 9, 14–16, 18, 22, 23, 25, 28, 29, 31, 34, 39, 40, 42–44, 48]. The knowledge base for our team's RAG solutions is our own product documentation, which is structured differently from Wikipedia articles.

Using common benchmarks to test your RAG implementation involves these steps: 1) index the given knowledge base content in your retriever component; 2) prompt your solution to answer the given questions; and 3) compare generated answers to expected answers, using methods such as exact match, cosine similarity, BLEU, ROUGE, METEOR, BertScore, or using LLMs as judges[49]. Those evaluation metrics have not been useful for evaluating our RAG results for novel questions from real users.

In this paper, we share our experience building enterprise-scale RAG solutions, with a focus on three aspects:

- **RAG implementation** – Our solutions are modular. Our retriever and generative components are closed boxes, accessed through APIs, with a limited ability to fine-tune.
- **Knowledge base content** – We are able to improve results by optimizing the knowledge base content itself. We developed content strategy and writing guidelines for RAG.
- **Evaluating results** – We test our RAG solutions with real user questions before making the solutions available to external users. We evaluate run-time answers after the solutions are launched.

Supporting material for this paper is available on GitHub.[1]

[1] https://github.com/spackows/ICAAI-2024_RAG-CD

2 RELATED WORK

Many techniques have been proposed for improving upon the basic RAG method. Advanced RAG techniques include: augment knowledge base content (with metadata or knowledge graphs, for example); fine-tune the embedding model, the retriever, or the generative model (or all of them); rewrite or expand the query; use multiple knowledge bases (diverse in their format and content) and then route queries; evaluate, re-rank, filter, and post-process retrieved chunks; generate multiple outputs to choose the best one; or iteratively refine results[4, 7, 11, 17, 26, 36–38, 45, 47, 50]. Little attention has been paid to optimizing the knowledge base content itself.

RAG solution builders must consider the structure of their knowledge base content when converting that content to text before indexing it for search. Much work has been done exploring the best way to read PDF documents, capture meaning from HTML or Markdown elements, interpret images, and reflect relational information in tables and lists[2, 3, 24, 30].

The structure of knowledge base content must also be considered when segmenting the content into chunks. Chunking too small risks splitting information across multiple chunks. Chunking too large risks including irrelevant information in a given chunk. Choosing a chunk size depends on multiple factors, including the profile of the knowledge base content[1, 3]. One way to include a complete, self-contained idea or explanation in each chunk is to chunk content not based on size, but at the chapter or section level[5].

The dominant search method in early RAG literature has been vector embeddings. Now, combining multiple search strategies (including traditional ones[2]) is increasing[19, 32].

When evaluating results from a deployed RAG solution, multiple authors have acknowledged manual work is required[10, 33, 49]. To assist with human evaluation, [27] and [6] propose identifying facts in questions, retrieved knowledge base content, and generated answers to confirm facts agree. ARES[35] and RAGAS[9] validate question-context relevance, context-answer faithfulness, and question-answer relevance. When evaluating their RAG solutions, [1] identify seven points of failure: 1) missing content; 2) search failure; 3) context window limitations; 4) poor answer generated by the LLM; 5) incorrect output format; 6) vague answers; and 7) incomplete answers. We can confirm seeing our solutions encounter those same failure points too.

3 RAG IMPLEMENTATION

User interface

Fig. 1 shows the interface of a simple RAG solution deployed on the search page of product documentation. A discussion of key user experience design aspects follows.

Search enhancement – When a search query is in the form of a natural language question, the usual search results are returned and a brief answer is generated by an LLM. Readers are accustomed to using the search bar to look for information, so there is no new interface to discover and learn how to use.

Not a chatbot – Previous dialog turns are not included in the LLM prompt for context. We chose to deploy a simple solution to get experience and feedback quickly. Also, our team is interested to explore non-chatbot LLM interfaces.

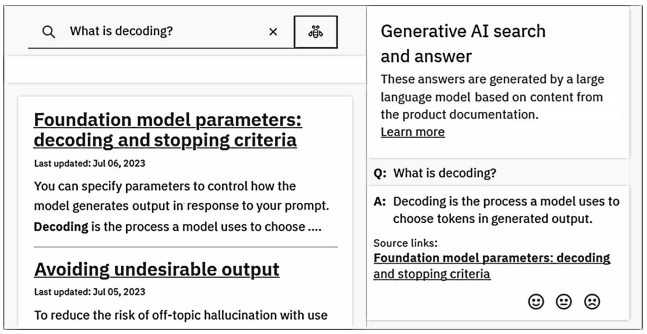

Figure 1: Search-and-answer user interface

Shaping user behavior – The search bar is a single line input, so it is awkward to type a complex question there. This friction nudges users to keep their questions concise. Fewer than 6% of questions submitted to the solution are much longer than the search input or require multi-hop reasoning. We plan to study the impact of the restricted input on that behavior.

Transparency and explainability – Links to content in which an answer is grounded are always provided. Also, terms that significantly impacted the way the solution generated the answer are highlighted in bold. When we review solution logs, we see the highlighting helps users know which terms to change (or remove) in their question to get a different answer.

Growth of natural language questions – Over time, the percent of queries submitted in the search bar that are expressed as a natural language question (versus a keyword search) has increased from 25% to 39%. (Fig. 2)

User feedback – We worried users wouldn't give feedback if it required multiple clicks, choosing from difficult-to-interpret categories, or typing explanations. But we also worried simple "thumbs-up" or "thumbs down" feedback wouldn't be fine-grained enough to analyze the impact of iterative solution improvements. So, we chose a 1-click interface with three options: "helpful", "somewhat helpful", and "unhelpful". We found that users give feedback less than 1% of the time, and mostly for unhelpful answers. (Fig. 3)

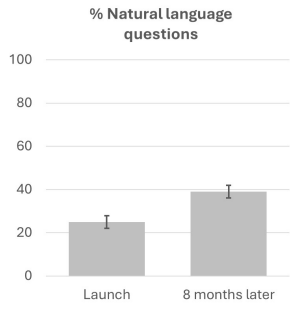

Figure 2

Figure 3

[2]https://applied-llms.org/#dont-forget-keyword-search-use-it-as-a-baseline-and-in-hybrid-search

Accessibility – Accessibility guidelines for static content have existed for some time.[3] And tools for testing HTML pages are readily available. But making LLM-driven natural language interfaces accessible has many open questions:

- Will generated text be clear for all reading levels?
- Can you bookmark a chat if you have memory challenges?
- Will generated images be clear if you see colors differently?
- Can you easily navigate generated output using a keyboard?
- Will generated video have captions and scene descriptions?

Solution architecture

Fig. 4 shows components of the RAG solution mentioned above. The knowledge base is product documentation made up of "topics", using the Darwin Information Typing Architecture (DITA) paradigm.[4] A discussion of key aspects follows.

(A) Pre-processing user input – Malicious input, such as JavaScript injection and adversarial prompts, is rejected; personal information removed; bias as well as hate, abuse, and profanity (HAP) paraphrased. Input is translated to English and classified to determine if it is a question and to detect the question type, such as "what-is", "how-to", or "troubleshooting". Only unharmful questions move on through the solution – in English.

(B) Frequently asked question (FAQs) – If a user's question matches a sensitive FAQ – related to legal terms, for example – we return a hard-coded answer. For other FAQs, we curate previously generated answers evaluated as useful. Before returning a curated answer, we confirm the grounding topics have not been updated, because that could change the answer. If the topics have been updated or deleted, the question is handled like a novel question.

(C) Augmenting the question – If the user's question does not match an FAQ, the question is further processed to improve search performance: ambiguous questions rewritten; jargon replaced with in-domain terms; synonyms added.

(D) Search as a closed box – We call a search API that returns a ranked list of topics relevant to our query. Some search results might be re-ranked or filtered by our RAG solution. A separate team manages the search API we use. They maintain the API and automatically index our documentation continuously.

(E) Whole topics instead of chunks – Once we have a list of relevant topics, we extract the complete text of those topics to ground our prompt. Sometimes called "small2big" or "parent document retrieval", this strategy works well for us because our topics are optimized for RAG. They are short, complete, accurate, and up-to-date. Our text-extraction component takes advantage of the reliable structure of our topics. For example, we have writing guidelines requiring tables to be fairly simple. Knowing this, we convert tables to row-wise and column-wise lists of lists to retain row-column relationships without worrying about complex tables.

(F) Simple prompts – Our solution isn't a dialog, so we don't need to maintain chat history. Because the user interface encourages simple questions, because we clean, clarify, and augment questions, because topics are optimized for RAG, and because we want concise answers that are faithful to the topics, the LLM has only one job: rewrite content from the grounding topics in a succinct answer. We prompt multiple models and choose the best answer.

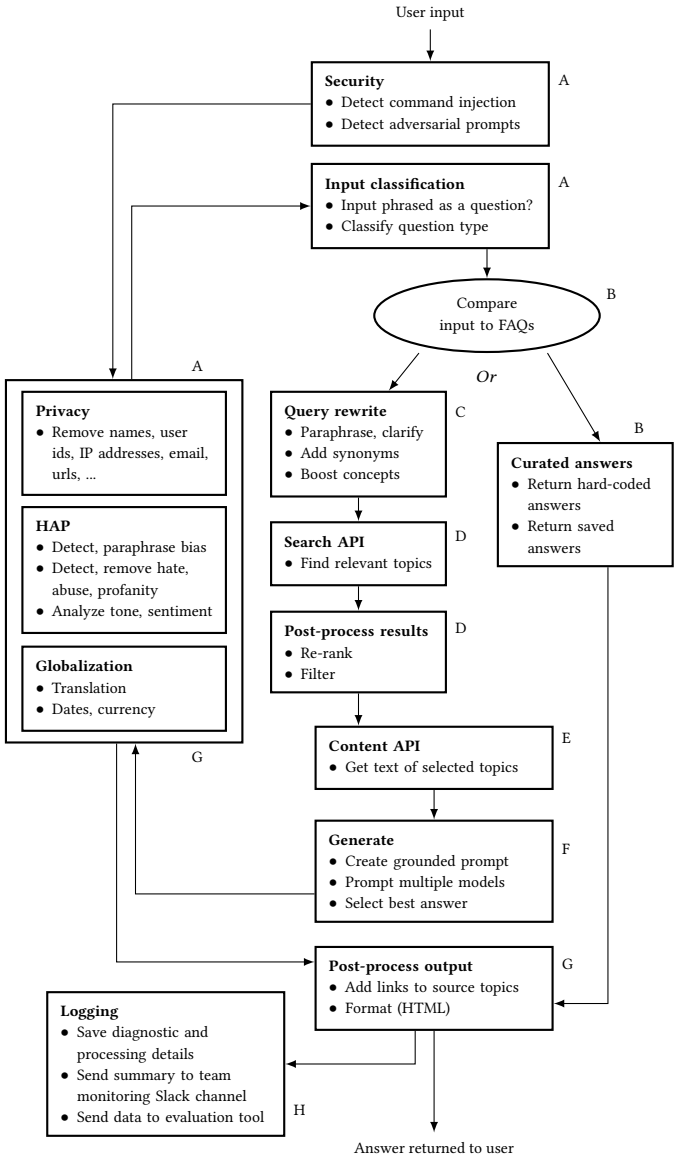

Figure 4: RAG solution diagram

(G) Post-processing output – Generated output is processed like the input: personal information, bias, and HAP generated by the LLM is removed or paraphrased. The answer is translated to the language of the original user question, links to grounding topics are added, and the whole output is marked up in HTML.

(H) Logging – Because LLMs can generate problematic output, our team monitors solution activity in real time by sending detailed logging to a team Slack[5] channel. We also send details to an evaluation tool discussed later in this paper.

[3] https://www.w3.org/WAI/standards-guidelines/wcag
[4] https://dita-lang.org/1.3/dita/archspec/base/introduction-to-dita
[5] https://slack.com

4 KNOWLEDGE BASE CONTENT

Our team uses search and LLM APIs that we have limited ability to adjust. What we can control is our knowledge base content. Our content must be easy to search, navigate, and consume by all readers, including people not working in their first language and people using tools like screen readers. Now, we also want our content to work well for RAG solutions.

Content rewriting experiment

We built a simple RAG solution to answer 189 questions about *the Earth* from the Natural Questions benchmark[22]. Initially, our solution did not answer all questions correctly.

One question we failed to answer is: "what is the pre-industrial level of co2 on earth?" The correct answer is: "280 ppm." But our solution responded with: "180 ppm."

Text from the relevant article used to answer that question follows. The underlined text is the only edit needed for the RAG solution to answer correctly:

> Over the past 400,000 years, CO2 concentrations have shown several cycles of variation from about 180 parts per million during the deep glaciations of the Holocene and Pleistocene to 280 parts per million during the interglacial periods until the pre-industrial era.

Minor edits like this increased success to 100%. Complete code and edits are available on GitHub.[a]

[a]https://github.com/spackows/ICAAI-2024_RAG-CD

For many RAG projects that use legacy knowledge base content, rewriting that content isn't feasible. However, for a RAG solution that is to be built 6 months from now or a year from now, the knowledge base content might not yet exist. Some estimate that more than 250,000 websites are created every day.[6] On Wikipedia, more than 400 articles are being added every day.[7] For our teams, products that will be released next year don't have any documentation yet. When creating new content, it makes sense to optimize it for RAG solutions.

Content strategy for RAG

Testing RAG solutions before making them available to users might seem difficult due to a lack of test questions that reflect what real users will ask[1]. However, when creating documentation for a new product or feature, content designers have always researched what questions users are likely to ask:

- We run internal workshops with teammates and observe where participants get stuck and what questions they ask.
- We read internal communities where teammates ask questions as they use internal releases of upcoming features.
- We review external forums where users are asking questions about similar functionality in other products.
- When features are in early, limited release, we collaborate with sales, pre-sales technical support, and customer advocates to find out what questions early users have.

These are all ways to collect questions that better represent what real users will ask than any questions we might guess ourselves. Optimizing content to be used in RAG solutions requires paying more attention to what questions that content must answer. A new quality metric will be: How well does a given topic answer anticipated user questions?

Testing topics

Imagine you have a list of real user questions about credentials and you have a draft topic about credentials. How could you verify the topic answers those questions?

You could prompt an LLM to answer the user questions grounded in the draft topic, then verify the answers. Or you could prompt an LLM to generate questions answered by the draft topic, then automatically compare the generated questions to the real questions. Hypothetical draft topic:

> Credentials are the user ID and password for authenticating with the service. Credentials are important, they prevent others using your service instance.

Hypothetical generated questions:

- What are credentials?
- Why are credentials important?
- What do credentials prevent?

The topic seems helpful and those seem like questions people might ask. But what if real user questions are: "Where do I find my credentials?", "How do I get my credentials?", and "Where can I look up my credentials?" The generated questions don't match those, which means a RAG solution using that topic won't work for real users.

Content guidelines for RAG

As we monitored our RAG solution results and then experimented with rewriting content, we identified patterns that led to better results. We asked writers from other teams to test these patterns with their content too. From this testing, we created guidelines to help writers optimize content for RAG:

- **Simplify complex tables** – Tables that have spanned cells or lack column headings are difficult for LLMs to interpret.
- **Explain graphics in text** – Explaining graphics clarifies ambiguities and avoids the need for an image-to-text model.
- **Add summaries to tutorials or long procedures** – LLMs struggle with long tutorials or procedures (because of getting "lost in the middle" or context window limitations.) Adding a summary is an easy way to improve results.
- **Clearly introduce lists** – LLMs can better use content in lists when there is a clear lead-in sentence before the list.
- **Simplify nested content** – Meaning can be lost by the LLM when content has multiple levels of nesting (steps with sub-steps that have option lists, for example.) Avoiding multi-level nesting improves results.

[6]https://www.forbes.com/advisor/business/software/website-statistics
[7]https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

5 EVALUATING RAG RESULTS

We created a web app that streamlines the task of manually reviewing and evaluating answers our RAG solutions return to users. Fig. 5 shows the evaluation page of the app. On the left is the user's question as well as some metadata, such as the language in which the question was submitted and the question classification. In the middle is the answer that was returned to the user, complete with generated answer text and links to relevant topics. On the right is a list of criteria:

- *Valid question* - Should we be able to answer this question?
- *Correct class* - Was the question classified correctly?
- *Article exists* - Is there a topic to answer the question?
- *Search success* - Did search find the relevant topics?
- *Good answer* - Is the answer accurate, complete, helpful?

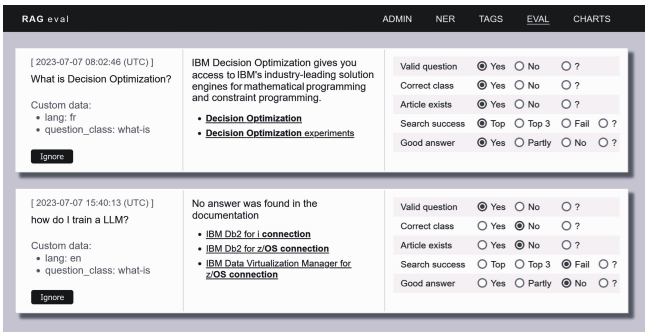

Figure 5: RAG evaluation web app

As we evaluate results, annotate key terms, and tag results, we end up naturally creating fine-tuning data sets, custom NLP dictionaries, and training data for classifiers. Our evaluation tool takes a "human in the lead" approach: AI learns from the data our manual work naturally creates so it can automatically perform some evaluations, entity identification, and classification.

Fig. 6 shows evaluation results for a RAG solution at two points in time. In July, for 40% of valid questions there were no topics containing information to answer the question. So, we recruited writers to fill that content gap. By December, there were topics to answer valid questions 75% of the time - an improvement. Unfortunately, search performance declined. In December, search didn't find the relevant topic 47% of the time. We were able to collect sample answers evaluated as `"Article exists" == "Yes"` and `"Search success" == "Fail"` so we could identify and fix the cause of the search failures. The RAG evaluation tool helps us know where to focus our improvement efforts.

Unanticipated benefits

Methodical evaluation of results has increased the business value of our RAG solutions:

- **Training data** - Sample questions, fine-tuning data sets, custom NLP dictionaries, and training data for classifiers naturally created as we manually evaluate, annotate, and tag results can sometimes be used to improve other AI solutions, such as analysing customer feedback surveys or community questions (subject to terms of use and reuse.)

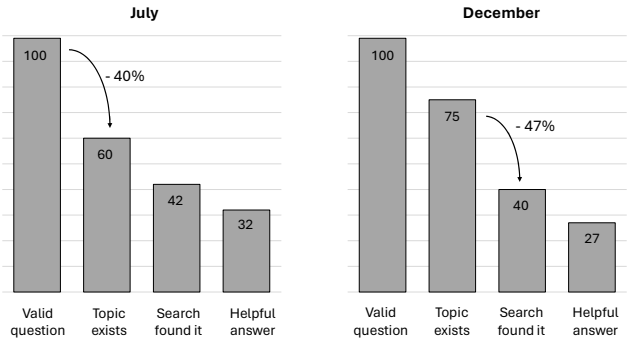

Figure 6: RAG evaluation results analysis

- **Insights** - The RAG evaluation tool sends a weekly summary of user questions to a team Slack channel so everyone knows what our users are struggling with and asking about.
- **Documentation improvements** - Our team meets for 30 minutes a week to review results and fix problems, including: content gaps, search failures, and content that needs editing.

6 SCALING AN ENTERPRISE SOLUTION

When building a RAG solution to support a portfolio of dozens or hundreds of software products, new challenges arise:

Questions vary by product - While common questions for one product might be factual "what-is" questions, common questions for another might be command-line syntax questions. A given question rewriting method might work for one but not the other.

Content varies by product - The documentation for one product might be conceptual or task-based, while another product's documentation might be mostly API reference details. Search that works well for one might not work well for the other.

Getting buy-in - When one product team decides to build a RAG solution, they feel invested and prepared to do manual work like evaluating results. But getting buy-in for an enterprise-wide initiative can be challenging.

One size might not fit all - Questions and content are not the only things that will vary across teams. Building one solution for everyone maximizes shared infrastructure. Ensuring that solution is configurable and flexible empowers individual teams to benefit from the centralized infrastructure while also doing what works best for them.

Automated regression testing - As teams rewrite their content and update components of their RAG solution, they need a way to test the performance of their solution without having to manually evaluate those test results. Solutions like the RAG evaluation tool can be used to collect question-topic-answer triplets that can be tested in automated batches. (Evaluation techniques like BLEU, ROUGE, and so on, *do* have a useful place here.)

ACKNOWLEDGEMENT

For supporting our work on these projects, we want to express our appreciation to our managers: Richard Horsfall, Wendy Switzer, Kirti Gani, and Lindsay Martin. Thank you!

REFERENCES

[1] Scott Barnett, Stefanus Kurniawan, Srikanth Thudumu, Zach Brannelly, and Mohamed Abdelrazek. 2024. Seven Failure Points When Engineering a Retrieval Augmented Generation System. arXiv:2401.05856

[2] Lukas Blecher, Guillem Cucurull, Thomas Scialom, and Robert Stojnic. 2023. Nougat: Neural Optical Understanding for Academic Documents. arXiv:2308.13418

[3] Andrei-Laurentiu Bornea, Fadhel Ayed, Antonio De Domenico, Nicola Piovesan, and Ali Maatouk. 2024. Telco-RAG: Navigating the Challenges of Retrieval-Augmented Language Models for Telecommunications. arXiv:2404.15939

[4] Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. RQ-RAG: Learning to Refine Queries for Retrieval Augmented Generation. arXiv:2404.00610

[5] Xinyue Chen, Pengyu Gao, Jiangjiang Song, and Xiaoyang Tan. 2024. HiQA: A Hierarchical Contextual Augmentation RAG for Massive Documents QA. arXiv:2402.01767

[6] I-Chun Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, and Pengfei Liu. 2023. FacTool: Factuality Detection in Generative AI – A Tool Augmented Framework for Multi-Task and Multi-Domain Scenarios. arXiv:2307.13528

[7] Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. 2024. The Power of Noise: Redefining Retrieval for RAG Systems. arXiv:2401.14887

[8] Emily Dinan, Stephen Roller, Kurt Shuster, Angela Fan, Michael Auli, and Jason Weston. 2019. Wizard of Wikipedia: Knowledge-Powered Conversational agents. arXiv:1811.01241

[9] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. 2023. RAGAS: Automated Evaluation of Retrieval Augmented Generation. arXiv:2309.15217

[10] Kshitij Fadnis, Siva Sankalp Patel, Odellia Boni, Yannis Katsis, Sara Rosenthal, Benjamin Sznajder, and Marina Danilevsky. 2024. InspectorRAGet: An Introspection Platform for RAG Evaluation. arXiv:2404.17347

[11] Masoomali Fatehkia, Ji Kim Lucas, and Sanjay Chawla. 2024. T-RAG: Lessons from the LLM Trenches. arXiv:2402.07483

[12] Philip Feldman. James R. Foulds and Shimei Pan. 2024. RAGged Edges: The Double-Edged Sword of Retrieval-Augmented Chatbots. arXiv:2403.01193

[13] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997

[14] Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies. arXiv:2101.02235

[15] Hiroaki Hayashi, Prashant Budania, Peng Wang, Chris Ackerson, Raj Neervannan, and Graham Neubig. 2020. WikiAsp: A Dataset for Multi-domain Aspect-based Summarization. arXiv:2011.07832

[16] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing A Multi-hop QA Dataset for Comprehensive Evaluation of Reasoning Steps. arXiv:2011.01060

[17] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active Retrieval Augmented Generation. arXiv:2305.06983

[18] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. arXiv:1705.03551

[19] Bongsu Kang, Jundong Kim, Tae-Rim Yun, and Chang-Eop Kim. 2024. Prompt-RAG: Pioneering Vector Embedding-Free Retrieval-Augmented Generation in Niche Domains, Exemplified by Korean Medicine. arXiv:2401.11246

[20] Mintong Kang, Nezihe Merve Gürel, Ning Yu, Dawn Song, and Bo Li. 2024. C-RAG: Certified Generation Risks for Retrieval-Augmented Language Models. arXiv:2402.03181

[21] Richard Khoury. 2009. The impact of wikipedia on scientific research. *Proceedings of the 3rd International Conference on Internet Technologies and Applications, ITA 09* (01 2009), 2–11.

[22] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: a Benchmark for Question Answering Research. *Transactions of the Association of Computational Linguistics* (2019).

[23] Remi Lebret, David Grangier, and Michael Auli. 2016. Neural Text Generation from Structured Data with Application to the Biography Domain. arXiv:1603.07771

[24] Demiao Lin. 2024. Revolutionizing Retrieval-Augmented Generation with Enhanced PDF Structure Recognition. arXiv:2401.12599

[25] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories. arXiv:2212.10511

[26] Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021. Generation-Augmented Retrieval for Open-domain Question Answering. arXiv:2009.08553

[27] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained Atomic Evaluation of Factual Precision in Long Form Text Generation. arXiv:2305.14251

[28] Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2020. AmbigQA: Answering Ambiguous Open-domain Questions. arXiv:2004.10645

[29] Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Mutethia Mutuma, Yasin Tarabar, Ankit Gupta, Tao Yu, Yi Chern Tan, Xi Victoria Lin, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. 2021. DART: Open-Domain Structured Data Record to Text Generation. arXiv:2007.02871

[30] Feifei Pan, Mustafa Canim, Michael Glass, Alfio Gliozzo, and James Hendler. 2022. End-to-End Table Question Answering via Retrieval-Augmented Generation. arXiv:2203.16714

[31] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. 2021. KILT: a Benchmark for Knowledge Intensive Language Tasks. arXiv:2009.02252

[32] Anupam Purwar and Rahul Sundar. 2023. Keyword Augmented Retrieval: Novel framework for Information Retrieval integrated with speech interface. arXiv:2310.04205

[33] Zackary Rackauckas. 2024. Rag-Fusion: A New Take on Retrieval Augmented Generation. *International Journal on Natural Language Computing* 13, 1 (Feb. 2024), 37–47. https://doi.org/10.5121/ijnlc.2024.13103

[34] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. arXiv:1606.05250

[35] Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. 2024. ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems. arXiv:2311.09476

[36] Alireza Salemi and Hamed Zamani. 2024. Evaluating Retrieval Quality in Retrieval-Augmented Generation. arXiv:2404.13781

[37] Kunal Sawarkar, Abhilasha Mangal, and Shivam Raj Solanki. 2024. Blended RAG: Improving RAG (Retriever-Augmented Generation) Accuracy with Semantic Search and Hybrid Query-Based Retrievers. arXiv:2404.07220

[38] Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. 2022. Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering. arXiv:2210.02627

[39] Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. 2023. ASQA: Factoid Questions Meet Long-Form Answers. arXiv:2204.06092

[40] Nandan Thakur, Luiz Bonifacio, Xinyu Zhang, Odunayo Ogundepo, Ehsan Kamalloo, David Alfonso-Hermelo, Xiaoguang Li, Qun Liu, Boxing Chen, Mehdi Rezagholizadeh, and Jimmy Lin. 2024. NoMIRACL: Knowing When You Don't Know for Robust Multilingual Retrieval-Augmented Generation. arXiv:2312.11361

[41] Neil Thompson and Douglas Hanley. 2018. Science Is Shaped by Wikipedia: Evidence From a Randomized Control Trial. https://doi.org/10.2139/ssrn.3039505

[42] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for Fact Extraction and VERification. arXiv:1803.05355

[43] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. arXiv:2108.00573

[44] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2020. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. arXiv:1905.00537

[45] Zhiruo Wang, Jun Araki, Zhengbao Jiang, Md Rizwan Parvez, and Graham Neubig. 2023. Learning to Filter Context for Retrieval-Augmented Generation. arXiv:2311.08377

[46] Kevin Wu, Eric Wu, and James Zou. 2024. How faithful are RAG models? Quantifying the tug-of-war between RAG and LLMs' internal prior. arXiv:2404.10198

[47] Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024. Corrective Retrieval Augmented Generation. arXiv:2401.15884

[48] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. arXiv:1809.09600

[49] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, and Zhaofeng Liu. 2024. Evaluation of Retrieval-Augmented Generation: A Survey. arXiv:2405.07437

[50] Tianjun Zhang, Shishir G. Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E. Gonzalez. 2024. RAFT: Adapting Language Model to Domain Specific RAG. arXiv:2403.10131