Advanced Retrieval Augmented Generation for Local LLMs

Leonardo Marques Rocha[1], Rian Manoel Pessoa[2]

[1,2] Plataforma de Inovação em Computação Cognitiva - Instituto Atlântico

Av. Washington Soares, 909, Fortaleza, 60811-341

Abstract. *This paper presents an extension of Retrieval Augmented Generation workflow for Large Language Models executing in a local processor with limited resources. The novelty presented is an improvement on such workflows to consider limitations in the total budget of token usage and privacy of data using local storage of data. This has the potential to leverage applications without the necessity of online services that can have a high cost and latency due the Chain of Thought used in most data retrieval cases. The presented workflow has a lightweight usage of computing and can be fully implemented in a low-resource compute environment.*

Resumo. *Este artigo apresenta uma extensão do fluxo de trabalho de geração aumentada de recuperação para Modelos de Linguagem Massivos executados em um processador local com recursos limitados. A novidade apresentada é uma melhoria em tais fluxos de trabalho para considerar limitações no orçamento total de uso de tokens e privacidade de dados utilizando armazenamento local de dados. Isto tem o potencial de alavancar aplicações sem necessidade de serviços online que podem ter alto custo e latência devido à Cadeia de Pensamento utilizada nos casos mais comuns de recuperação de dados. O fluxo de trabalho apresentado tem um uso leve de computação e pode ser totalmente implementado em um ambiente de computação com poucos recursos.*

1. Introduction

The field of Artificial Intelligence (AI) has been overflowing with new technology and heuristics in the past 5 years since the popularization and major adoption of the Transformer architecture for Generative AI. (GenAI). Although the overall success of GenAI as a frontier technology, in essence its major advantage is to produce content and emergency capabilities, as mentioned by Wei et al. (2022) and Bubeck, 2023, and can learn long-horizon behaviors by latent imagination from dream-like representation in the latent layers, Ha and Schmidhuber (2018). Nevertheless, this is an adversary behavior for generating factual outpurs, and there is such a case, the term “hallucination” has been used to describe it. Although a lot of improvements with Large Language Models (LLM) has been archived with LLM such as GPT-4o, from OpenAI, hallucination is a natural characteristic as mentioned by one of its founders in social media, Karpathy, A. (2023, August 1): “I always struggle a bit with I'm asked about the "hallucination problem" in LLMs. Because, in some sense, hallucination is all LLMs do. They are dream machines. It's only when the dreams go into deemed factually incorrect

territory that we label it a "hallucination". It looks like a bug, but it's just the LLM doing what it always does. (...)".

If the previous citation is continued it is possible to understand the need for better guidance of LLMs: "(...) We direct their dreams with prompts. The prompts start the dream, and based on the LLM's hazy recollection of its training documents, most of the time the result goes someplace useful.". Some applications cannot suffer from hallucination as part of the process and, although some guidance can be done using model refinement, or by using In Context Learning (ICL), Radford et al. (2019), such as one-shot or few-shot, neither can prevent cases of hallucination. Another usage for the latter is to provide new content to the LLM that has not been trained in its base model. Many techniques have been proposed to iterate in cycles to align prompts and outputs, such Chain Of Thoughts (CoT), as in Wei et al. (2022), but they do not use any information that is already in the prompt or in the latent space of the LLM. Retrieval Augmentation Generation (RAG) has been proposed in Lewis et al. (2020) to gather additional information to improve the generation, in a single-shot manner, provided as context to the prompt. Because of the limited size of the context window in the Transformer architecture, the information needs to be splitted in smaller chunks to fit the context window of the LLM, or summarized, or encoded such in a way that can be used in a caching system.

2. Proposed work and Motivation

This paper presents a new workflow that specifically tries to address some limitations in RAG systems: hallucinations due the prompt engineering does not align with the expected answer. Such workflows are not new, and are being used with much deeper and greater in number of parameters available on online services such GTP-4o, Claude Opus, and others. LLMs available for public use can have up to 405 billions of parameters, such as the latest Llama and its variants, and if the process uses surge large LLM, the penalty comes with latency and expensive hardware. While not mandatory, for practical use it is advisable to use the LLM in an accelerated computing environment, usually with GPUs with large dedicated RAM space to fit the model and compute artifacts to mitigate latency. Also, not all LLMs have the same capabilities, some more suitable for using external tools that can fetch data during the autoregressive process, and adding custom context on-demand for a better inference, This has been the reason for dedicated online services that can provide some functionality with tools and reasoning the output for a better prompt. This comes with a total cost as the number of tokens increases exponentially, because now each part of the prompt has to be reflected in one or more steps, branching in sub-tasks that are later aggregated to create a prompt with the necessary, and factual, context.

While hallucination is part of what a LLM usually does as mentioned before, it has to be steered to focus on the task at hand, hedging the risks of wrong interpretability of intention with the usage of correct references to base its answers. While basic RAG workflows are more concerned with the retrieval and storage of documents, more advanced techniques incorporate CoT on the retrieval loop to fetch better context. For the generation of better reasoning CoT, online services with top-level LLMs are used.

This is a major problem for the unprivileged majority with modest computing resources and short budget limitations. For those that do not have access to such resources, the usage of smaller, quantized, but yet capable, LLMs renders another possibility to use such workflows within such limited budget cost. While with the restriction to fit and run in local computer hardware has been lifted for practitioners and researchers that are developing possible workflows for dedicated problems.

The result of this work is a workflow that can be customized for dedicated problems without losing its generalizability. It combines a couple of RAG strategies (Gao, et al. (2023) to create an advanced RAG that can answer a question, or just avoid the hallucinations when not possible, returning a message to the user that is not possible to answer based on the factual information available that has been given or fetched with tools.

3. Related work

This section presents an overview of key works that have shaped the understanding and development of Generative AI, with a focus on Transformers, emergent capabilities, and techniques to mitigate issues like hallucination. Each reference contributes to a specific aspect of the challenges and advancements in the field. Below are the key topics mentioned and the corresponding references:

1. Transformers and Generative AI: The transformative impact of the Transformer architecture has been well-documented since its introduction by Vaswani et al. (2017), which paved the way for Generative AI models such as GPT-3, discussed extensively by Brown et al. (2020).

2 Emergent Capabilities in AI: The unexpected emergent capabilities of large language models were thoroughly investigated by Wei et al. (2022), demonstrating that as model sizes increase, novel behaviors and capabilities emerge. Also the pape

3. Dream-like Representations and Latent Imagination: The concept of models "dreaming" or imagining scenarios within a latent space is explored by Ha and Schmidhuber (2018) in their work on World Models.

4. Hallucinations in LLMs: The issue of hallucinations in language models, where the model generates factually incorrect or nonsensical outputs, is discussed comprehensively by Ji et al. (2023). Additionally, Andrej Karpathy's commentary on X, Karpathy, A. (2023, August 1), provides a more personal and practical perspective on this phenomenon.

5. In-Context Learning (ICL) and Few-Shot Learning: The use of In-Context Learning to adapt models to new tasks with minimal examples is exemplified by Radford et al. (2019), showing its importance in the broader context of few-shot learning.

6. Chain of Thought (CoT) and Prompt Engineering: The strategy of eliciting reasoning through Chain of Thought prompting, as discussed by Wei et al. (2022), highlights the importance of well-structured prompts in guiding model outputs.

7. Retrieval Augmented Generation (RAG): Lewis et al. (2020) introduced the Retrieval-Augmented Generation technique, which enhances the factuality of generated text by integrating external knowledge during generation.

8. Quantized LLMs and Model Deployment: The deployment of large models on limited hardware resources through quantization techniques is addressed by Dettmers et al. (2022), demonstrating the feasibility of running powerful models in resource-constrained environments.

9. Retrieval-Augmented Generation for Large Language Models: A Survey: Gao, et al. (2023) highlights the state-of-the-art technologies embedded in each of these critical components, providing a profound understanding of the advancements in RAG systems.

4. Proposed methodology

The proposed methodology is an extension of a RAG workflow that implements the Advanced RAG as described in Gao et al, (2024). This classification is a comparison with other model optimization methods in the aspects of "External Knowledge Required" and "Model Adaption Required", that gives two axes of: how much modifications to the model versus the external knowledge that are necessary to harness the capabilities of LLMs themselves. In this scale the Advanced RAG needs more external knowledge than Naive RAG, that is basic Prompt Engineering. Advanced RAG is classified as an index / pre-retrieval / post-retrieval optimization process. This retrieval augmentation processes adds to most common once retrieval process to create general types of processes, exactly as described as follows:

- Iterative retrieval: involves alternating between retrieval and generation, allowing for richer and more targeted context from the knowledge base at each step;
- Recursive retrieval: involves gradually refining the user query and breaking down the problem into sub-problems, then continuously solving complex problems through retrieval and generation;
- Adaptive retrieval: focuses on enabling the RAG system to autonomously determine whether external knowledge retrieval is necessary and when to stop retrieval and generation, often utilizing LLM-generated special tokens for control.

These classification groups are useful to identify the proposed methodology as an Adaptive retrieval that differs from CoT pipelines in solving complex problems. The concern of the proposed methodology is to have a small token budget as part of the process, and breaking it down into sub-problems creates an exponentially tree that is not the focus for application in limited resources and latency restrictions.

Some of those steps from Iterative retrieval and Adaptive retrieval are replicated in a different way in the proposed Advanced RAG, while the score is binary, based on the LLM inference of the context given and the prompt of each step, that are described in the Figure 1.

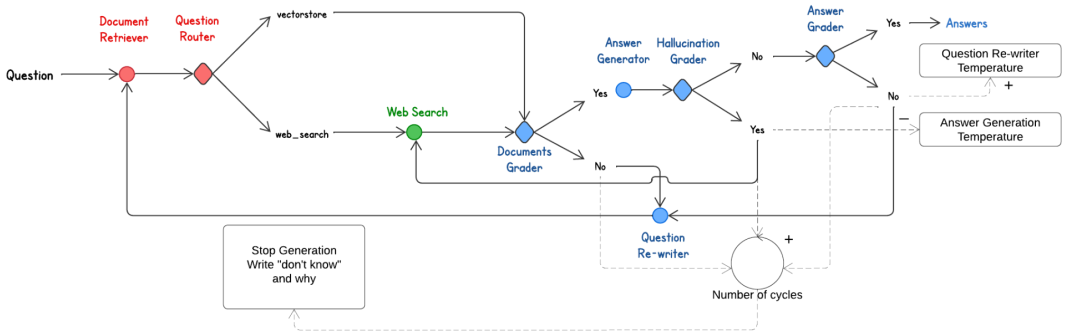

Figure 1. A schematic of the workflow (arrows) of the proposed RAG with nodes and actions (dotted arrows) annotated with brackets of each group by color: a) red for router retriever decision, b) green for retrieval using web search, and c) blue nodes branching for grading documents in regard of relevance of the question, and answer have presence of hallucination and answers the question. d) Actions to parameter change in boxes with positive and negative increases.

The workflow starts with the user question and checks if it has an index in the local document store as shown in the red node in Figure 1a. The Question Router node (represented by rounded diamond) uses a local LLM to analyze chunks and decide using a binary answer if the document should be retrieved from the datastore or be fetched online using web search. In each case the result is the same and it is graded by Document Grade node in accordance with the relevance to the question. A decision is taken, giving a binary score: yes or no. If yes, the process follows the upper path in Figure 1c and the answer is generated in the Answer Generator node. The generated answer is tested for the presence of hallucinations using another binary score: yes or no. If the result is negative, the process continues in the upper path to finally be scored again by the Answer Grader node if the generated answer is relevant to the question. The same binary scoring is used to decide whether it answers correctly the question, and return it as result.

The alternative branching in the previous path is intended to rewrite the question and return the generation to the Document Retriever node. There are additional steps if the branching is due hallucination or due relevance to the question. If there is hallucination the process goes back to the Web Search node, since the problem is lack of fact to support the answer In both cases, the branching increases the counter of cycles to limit the number of retries of the process and give an honest answer “don't know”. This is an acknowledgment of the natural process of hallucination or misguidance of the LLM, and the temperature of the model used to generate the answer is decreased. If the answer still does not answer the question, a last branching increases the temperature of the LLM, in the expectation that it rewrites the question with extra creativity in the Question Re-writer. This tries to avoid endless loops if the rewritten question is always the same. The opposite happens with hallucination branching, it lowers the temperature of the LLM in order to generate less creative answers, and lower the probability of generation of hallucinations.

5. Implementation and Results

Without loss of generality, the goal of this work is to use local and small models, but any other can be interchanged to use available computing resources at the best capacity, with greater latency and memory usage as disadvantage. Regarding that, the motivation is the opposite, to have better utilization of resources with almost the same token budget, as newer small LLMs are getting better performance and reasoning capabilities.

5.1. Implementation

The latest improvements of Llama 3 LLMs in its release shows better results over other LLMs with similar LLMs *w.r.t.* the number of parameters. This is achieved with better training with web data curation, text extraction and cleaning, deduplication, heuristic filtering, model-based quality filtering, code and reasoning data; to provide similar versions, Llama 3 7b and Llama 3.1 7b that differ mostly by tool usage. While Lamma3 7b has been post-trained to use tools such as search engines to expand the range of tasks, they both can generate an output encoded in JavaScript Object Notation (JSON).

The nodes of the Advanced RAG are implemented using LangChain Expression Language (LCEL) that not only creates templates for prompts, with system and user commands, but also chains the LLM and its output to other steps in the same node, structuring the output in JSON format that follows an desired schema given by a code. This output is already created only if the LLM supports the usage of tools such Llama 3.1. The previous version of the same model Llamma3 does not have this functionality, and the text output has to be parsed in the chain of the LCEL to generate a mapping of keys and values in the same structure as the JSON. For instance, in order to guide the LLM to JSON outputs the message is appended to the prompt:

“Provide the binary score as a JSON with a single key '{key}' and value either '{ok_msg}' or '{fail_msg}'.”

where {ok_msg} or {fail_msg} are value fields filled with particular node values for branching for a given {key}.

The Router node has a local storage with pre-fetched documents from the web in a vector store. The task of this node is to check the relevance of the question to each document using the system prompt:

“You are an expert at routing a user question to a vectorstore or web search.

Give a binary choice '{ok_msg}' or '{fail_msg}' based on the question.”

where {ok_msg} or {fail_msg} are “vectorstore” or “web_search”

If all documents are irrelevant to the question the decision branch takes the next step to the Web Search node, which performs a web search using top 4 results and stores the results in the vectorstore.

The Grade node assess the relevance of an retrieved document added to the prompt context using the system prompt:

"You are a grader assessing the relevance of a retrieved document to a user question.

If the document contains keyword(s) or semantic meaning related to the user question, grade it as relevant.

It does not need to be a stringent test. The goal is to filter out erroneous retrievals.

Give a binary score '{ok_msg}' or '{fail_msg}' score to indicate whether the document is relevant to the question."

where the binary choice is "yes" or "no", based on the assessment.

The Generate node answers the question and prompt checks for hallucination using the system prompt:

"You are a grader assessing whether a generated answer is grounded in or supported by a set of facts.

Give a binary score '{ok_msg}' or '{fail_msg}' score to indicate whether the document is relevant to the question."

where the binary choice is "no" or "yes", as it indicates the answer is supported by the facts given in the context.

The final decision, in case there is no hallucinations, or "no" in the previous step, is to address the answer as relevant to the question with the system prompt:

"You are a grader assessing whether an answer addresses / resolves a question.

Give a binary score '{ok_msg}' or '{fail_msg}' score to indicate whether the document is relevant to the question."

where the binary choice is again "yes" or "no", based on the assessment. The question can be not the original user question, but a rewritten version if there is hallucination or not relevant in the previous steps.

5.2. Results

This section shows the experiments for the workflow presented and implemented in the previous sections. They consist of 10 questions, Q1 to Q10 and the experiments used increasing difficulty grouping by 3 easy questions, 4 regular questions, 3 hard questions.

Table 1b shows Q1 to Q5 and the node output with the total number of items that were generated in the whole process to the final answer, while Table 1a shows the parameters used for each question, and both Tables 1a e 1b uses Llama 3 7b as model for all nodes. The model could answer all questions without iterating, hence 1 in all parentheses in the binary answers in Table 1b, but in most cases it used web search instead of using the vector store for answering the question at the first cycle.

Table 2b shows Q6 to Q10 and the node output with the total number of items that were generated in the whole process to the final answer, while Table 2a shows the parameters used for each question, that have increased chunk size and overlap for hard question Q8 to Q10, but Tables 2a e 2b still uses Llama 3 7b as model for all nodes. The cells with bold typing show an increase in iteration cycles for Q6 and Q7, going back to the retriever node, but in Q6 the branching happened earlier than Q7 in the document grader

node, thus a smaller overhead to get the correct answer than going over all nodes. The harder questions did not answer the questions with the same parameters of chunk size and overlap, and increasing them answered the question without any branching, but with a higher cost of tokens used as context.

Parameters	Model	Q1	Q2	Q3	Q4	Q5
Answer G. Temp.	Llama3	0.4	0.4	0.4	0.4	0.4
Rewriter G. Temp.	Llama3	0.0	0.0	0.0	0.0	0.0
Chunk Size	None	256	256	256	256	256
Chunk Overlap	None	2	2	2	2	2

Table 1a. Parameters for questions Q1 to Q5: final temperature and retriever chunk size and overlap between chunks

Process	Model	Q1	Q2	Q3	Q4	Q5
Retriever	None	doc(4)	doc(4)	doc(4)	doc(4)	doc(4)
Question Router	Llama3	web_search(1)	vectorstore(1)	web_search(1)	web_search(1)	web_search(1)
Web Search	None	doc(4)	None	doc(4)	doc(4)	doc(4)
Document Grader	Llama3	yes(1)	yes(1)	yes(1)	yes(1)	yes(1)
Answer Generator	Llama3	answer(1)	answer(1)	answer(1)	answer(1)	answer(1)
Hallucination Grader	Llama3	no(1)	no(1)	no(1)	no(1)	no(1)
Answer Grader	Llama3	yes(1)	yes(1)	yes(1)	yes(1)	yes(1)
Re-writer	Llama3	None	None	None	None	None

Table 1b. Outputs from the interaction processes (in bold) of each node for questions Q1 to Q5 with the number of outputs in the parentheses

Parameter	Model	Q6	Q7	Q8	Q9	Q10
Answer G. Temp.	Llama3	**0.3**	0.4	0.4	**0.3**	0.4
Rewriter G. Temp.	Llama3	0.1	0.0	0.0	0.0	0.0
Chunk Size	None	256	256	512	512	512
Chunk Overlap	None	2	2	10	10	10

Table 2a. Parameters for questions Q1 to Q5: final temperature and retriever chunk size and overlap between chunks

Process	Model	Q6	Q7	Q8	Q9	Q10

Retriever	None	doc(4)	doc(4)	doc(4)	doc(4)	doc(4)
Question Router	Llama3	**vectorstore (2)**	**vectorstore (2)**	web_search (1)	vectorstore(2)	web_search (1)
Web Search	None	doc(4)	doc(4)	doc(3)	doc(4)	doc(3)
Document Grader	Llama3	**yes(2)**	**yes(2)**	yes(1)	yes(1)	yes(1)
Answer Generator	Llama3	**answer(2)**	answer(1)	answer(1)	answer(1)	answer(1)
Hallucination Grader	Llama3	**no(2)**	no(1)	no(1)	no(1)	no(1)
Answer Grader	Llama3	**yes(2)**	yes(1)	yes(1)	yes(1)	yes(1)
Re-writer	Llama3	question(1)	question(1)	None	None	None

Table 2b. Outputs from the interaction processes (in bold) of each node for questions Q1 to Q5 with the number of outputs in the parentheses

The same workflow has been tested with Llama 3.1, with similar results in the execution of the isolated nodes. This section shows only the results of Llama 3 as unit testing the Llama 3.1 it outputs the same answers of Llama 3, but uses tool calling with structured output to a data model.

6. Conclusion

This paper presented a new Advanced RAG workflow that combines some aspects of different RAG systems that is dedicated to smaller models and applications with limited budget in tokens. The short communication between nodes and forcing the output format to a JSON creates a more robust system, as the outputs are in a structured format.

Exploration for Llama 3.1 has been done and all nodes have been unit tested with the native support for structured output mapped to a data object, but left for a future work with more space for discussion and implications of such methodology.

Another future subject is the exploration of evaluation using the last layer of the models, the logits distribution, as it can possibly be used to identify hallucinations.

7. References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998-6008).

2. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., & Amodei, D. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

3. Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, Q., Lester, B., Du, N., Dai, A. M., Smolensky, P., & Le, Q. (2022). Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

4. Ha, D., & Schmidhuber, J. (2018). World models. *arXiv preprint arXiv:1803.10122*.

5. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., & Bang, Y. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys (CSUR)*, 55(12), 1-38.

6. Karpathy, A. (2023, August 1). Personal communication on X (formerly Twitter). Retrieved from [X Platform](https://x.com).

7. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 9.

8. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., & Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

9. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., & Rocktäschel, T. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, 33, 9459-9474.

10. Dettmers, T., Lewis, M., Shleifer, S., & Zettlemoyer, L. (2022). LLM.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.

11. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2024). Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint arXiv:2312.10997*. Retrieved from https://arxiv.org/abs/2312.10997.

12. Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., & Zhang, Y. (2023). Sparks of Artificial General Intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*. Retrieved from https://arxiv.org/abs/2303.12712.