RAG EVALUATION

Christopher Vishnu Kumar

ABSTRACT

This technical report evaluates five different retrieval-augmented generation (RAG) implementations tested across 30 questions to gauge their ability to retrieve and generate information accurately. The evaluation employed a grading system categorising each response into pass, partial, or fail. Among the systems tested, OpenAI distinguished itself as the most effective, particularly in its API access mode, which scored 24 "passes" and 6 "partials", and its ChatGPT mode, which achieved 23 "passes" and seven "partials". Two of the three methods implemented using open-source models, while not successfully answering many questions, show potential having only partially answered user queries. The report acknowledges the potential for human error in evaluating responses, particularly where subjective judgment may influence the categorisation and subsequently affect the overall assessment results. The scripts used for testing as well as the gathered results can be found at https://github.com/christopherkumar/RAG-evaluation.git.

1. WHAT IS RETRIEVAL-AUGMENTED GENERATION?

Retrieval-augmented generation (RAG) refers to the integration of retrieval mechanisms with generative models, enhancing the model's ability to produce relevant and accurate content by incorporating external knowledge. This method has been developed in response to the limitations of traditional language models, which can struggle with accuracy and relevance when generating text based solely on learned patterns without access to external information.

One of the innovative approaches within this field is Adaptive Retrieval-Augmented Generation (ARAG), which dynamically determines the necessity of retrieval for queries, avoiding indiscriminate retrieval and thus enhancing the efficiency and relevance of the sourced information (Zhang, Fang & Chen, 2024). This approach addresses the absence of direct evaluation in previous works. It has been tested using the RetrievalQA benchmark, which includes a wide range of short-form questions that require external knowledge for accurate responses.

Another notable advancement is the Hybrid Retrieval-Augmented Generation (HybridRAG) framework, which combines a cloud-based large language model with a smaller, client-side model. This method facilitates real-time applications such as composition assistance, where it is crucial to maintain low latency without compromising the quality of the generated content (Zhang et al., 2023).

Furthermore, the Loops On Retrieval Augmented Generation (LoRAG) architecture introduces an iterative loop mechanism that refines the generated text through continuous interaction with the retrieved information. This model has shown superior performance over existing models, particularly in terms of coherence and relevance in the generated text (Thakur & Vashisth, 2024).

Lastly, the concept of Active Retrieval Augmented Generation employs a more dynamic approach, where the model actively decides when and what information to retrieve throughout the generation process. This method is particularly effective in generating long texts. It has been encapsulated in the Forward-Looking Active Retrieval augmented generation (FLARE) model, which anticipates future content needs and retrieves information accordingly to ensure factual accuracy (Jiang et al., 2023).

2. RAG IMPLEMENTATIONS

Five different RAG implementations were evaluated, three using open-source models and two using OpenAI's services. All resource versions are current as of 27 May 2024.

- Method 1: Adapted for testing with resources from https://github.com/pixegami/rag-tutorial-v2.git.
- Method 2: Adapted for testing with resources from https://github.com/AllAboutAI-YT/easy-local-rag.git.
- Method 3: Adapted for testing with resources from https://github.com/PromtEngineer/localGPT.git.
- Method 4: RAG implementation using OpenAI vector stores, assistants, and file search utilities.
- Method 5: RAG implantation using OpenAI's ChatGPT.

Methods 1 and 2 utilised Ollama's Python library for embedding and language model usage. `mxbai-embed-large` and `Mistral` were used for the embedding and large language model (LLM). The Ollama-Python documentation can be found at https://github.com/ollama/ollama-python.git.

Method 3 used HuggingFace's model repositories, using `mixedbread-ai/mxbai-embed-large-v1` and `TheBloke/Mistral-7B-Instruct-v0.2-GGUF/ mistral-7b-instruct-v0.2.Q8_0.gguf` for the embedding and language models.

Method 4 utilised OpenAI's Python library for API access. OpenAI's assistants ,vector store and file search tools were created and used to implement RAG functionality.

Method 5 employed OpenAI's ChatGPT, using gpt-4 (and gpt-4o when token limits were exceeded) and their custom GPT tool.

Although it is not clearly defined in the documentation, Method's 4 and 5 most likely use either `text-embeddings-3-small` or `text-embeddings-3-large`.

A more detailed outline can be found at the GitHub repository referenced in the Abstract.

3. RESULTS

The results and response evaluation table are listed in Appendix A.

TOTAL OUTCOMES ACROSS ALL METHODS

METHOD 1: MISTRAL WITH MXBAI EMBED-LARGE

- Pass: 13
- Partial: 11
- Fail: 6

METHOD 2: MISTRAL WITH MXBAI EMBED-LARGE

- Pass: 13
- Partial: 7
- Fail: 10

METHOD 3: MISTRAL-7B-INSTRUCT-V0.2 WITH MXBAI EMBED-LARGE V1

- Pass: 12
- Partial: 13
- Fail: 5

METHOD 4: GPT-4 WITH VECTOR STORE AND FILE SEARCH

- Pass: 24
- Partial: 6
- Fail: 0

METHOD 5: CUSTOM GPT

- Pass: 23
- Partial: 7
- Fail: 0

KEY OBSERVATIONS:

High Performance of GPT-4 Based Implementations: Both Method 4 and Method 5, which utilise some form of GPT-4, significantly outperformed the other methods in terms of passing grades, with no failures recorded. This indicates a strong ability of these implementations to generate accurate and relevant responses based on the source documents.

Issues with Fails in Mistral Implementations: Method 2 experienced the highest number of fails, which suggests issues with either the RAG implementation's document retrieval or processing, leading to incorrect or hallucinated data in responses.

Variability in Partials: Method 3 had a high number of partial passes, suggesting that while the responses were on the right track, they either lacked full information or included some incorrect elements. This could point towards the need for improvements in fine-tuning or a more refined retrieval component.

4. Positives and Drawbacks of Each Implementation

Method 1 - pixegami/rag-tutorial-v2

Positives:

- Simplifies the cataloguing of models, making management and accessibility straightforward.

Drawbacks:

- Testing is limited to q4-bit models, which may not fully represent performance at higher bit rates.

Method 2 - AllAboutAI-YT/easy-local-rag

Positives:

- Facilitates easy cataloguing and access to models, enhancing user experience and efficiency.

Drawbacks:

- Testing constrained to q4-bit models, potentially limiting comprehensive performance evaluation.
- Utilises chromaDB stored in memory, rendering it impractical for deployment due to high memory demands.

Method 3 - PromtEngineer/localGPT

Positives:

- Allows for flexible, local operations without the need for continuous internet connectivity.

Drawbacks:

- Dependent on the local device's specifications, which can restrict the models' operational capabilities:
- Storage: Only a limited number and size of models can be managed.
- Memory: Predominantly VRAM, with limitations affecting the feasibility of running 'f16/32' models. Models are typically limited to 'q8' on entry-level consumer cards with 8GB VRAM.
- The embedding model is maintained in memory, which could lead to significant storage space requirements.
- Limited to using models available through https://huggingface.co/TheBloke. (Unsure if this only occurred with regards to how I was implementing model changes)
- Encountered specific issues with llama cpp in how the chat template was implemented, impacting the performance of quantised model versions. Further details are available on GitHub at https://github.com/ggerganov/llama.cpp/issues/6747.

Method 4 - API Usage

Positives:

- Provides seamless integration and access to current OpenAI models through the API.

DRAWBACKS:

- The model architecture is subject to change as it is closed source, which can lead to unexpected alterations in functionality.
- API access is tiered, with higher rates and token limits available at premium tiers. For instance, Tier 5 necessitates a minimum of $1,000 payment and a waiting period of over 30 days since the first successful payment, with a usage cap of $15,000/month. This setup creates a significant overhead for certain use cases.
- Requires diligent monitoring to ensure that the usage limits or prepaid balances are not exceeded, demanding regular administrative attention.

METHOD 5 - CUSTOM GPT

POSITIVES:

- Each query allows for responses with extended token limits, offering more comprehensive and detailed outputs.

DRAWBACKS:

- As with other OpenAI models, the architecture is closed source and subject to modification without prior notice.
- Users lack control over the model selection, with the system defaulting to the best available model and switching to GPT-4o upon reaching token limits.
- Free usage of custom GPT models enforces stricter token rate limits and imposes restrictions on the selection of available models.
- File management constraints include:
  - A maximum of 20 files.
  - Individual file size not exceeding 512MB.
  - A cumulative size limit of 100GB for all uploaded files.
  - Each file can contain no more than 2,000,000 tokens.

5. SUMMARY

This technical report provides a comprehensive evaluation of five different retrieval-augmented generation (RAG) implementations, assessing their performance in accurately retrieving and generating responses across 30 questions. The tested methods encompass a mix of open-source models and OpenAI's proprietary models, each exhibiting distinct strengths and limitations as revealed through a rigorous pass, partial, and fail grading system.

KEY FINDINGS:

OpenAI's Implementations (Methods 4 and 5) demonstrated superior performance with the highest number of passes and no failures. This suggests that their models, which include GPT-4 with vector store, file search utilities, and Custom GPT capabilities, are robust and effective in generating relevant and accurate information.

Methods Utilising Open-Source Models (Methods 1, 2, and 3) showed varying levels of success. While they were able to address many questions partially, the number of fails, particularly with Method 2, indicates challenges in either the retrieval accuracy or the generation process. A potential occurrence was the LLM falling back to its own trained knowledge base to answer questions when it was unable to find relevant information from the embeddings.

Technical constraints and drawbacks varied across the methods, with local implementations potentially facing limitations due to device specifications and OpenAI's models being constrained by API access rules and closed-source architecture changes.

CONTRIBUTIONS:

The report highlights the feasibility and challenges of implementing RAG systems using both open-source tools and commercial APIs. It also underscores the importance of choosing the right model and configuration to balance performance with practical deployment considerations.

IMPLICATIONS FOR FUTURE RESEARCH:

The mixed results across different implementations suggest that further research could explore hybrid approaches that combine the strengths of open-source flexibility and the robust capabilities of commercial APIs. Additionally, optimising the configuration settings and improving error handling in open-source methods could enhance their reliability and output quality.

PRACTICAL RECOMMENDATIONS:

Relying on proven methods like OpenAI's implementations may offer more reliability and fewer operational risks for deployment in critical applications.

For developmental or experimental settings, open-source methods offer valuable learning opportunities and customisation options, albeit with a need for careful management of technical limitations.

Overall, this report provides valuable insights into the practical application of RAG technologies, guiding users in selecting appropriate methods for their specific needs and highlighting areas for further enhancement in RAG system implementations.

Resources

Python Documentation

Ollama, 2023, ollama-python, GitHub repository, viewed <viewed-date-here>, https://github.com/ollama/ollama-python.git.

OpenAI, 2023, openai-python, GitHub repository, viewed <viewed-date-here>, https://github.com/openai/openai-python.git.

Open-Source RAG Repositories

Pixegami, 2023, rag-tutorial-v2, GitHub repository, viewed <viewed-date-here>, https://github.com/pixegami/rag-tutorial-v2.git.

AllAboutAI-YT, 2023, easy-local-rag, GitHub repository, viewed <viewed-date-here>, https://github.com/AllAboutAI-YT/easy-local-rag.git.

Prompt Engineer, 2023, localGPT, GitHub repository, viewed <viewed-date-here>, https://github.com/PromtEngineer/localGPT.git.

HuggingFace Documentation

Mixedbread-AI, 2023, mxbai-embed-large-v1, Hugging Face, viewed <viewed-date-here>, https://huggingface.co/mixedbread-ai/mxbai-embed-large-v1.

TheBloke, 2023, Mistral-7B-Instruct-v0.2-GGUF, Hugging Face, viewed <viewed-date-here>, https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-GGUF.

OpenAI Documentation

OpenAI, 2023, Vector Stores API Reference, OpenAI, viewed <viewed-date-here>, https://platform.openai.com/docs/api-reference/vector-stores.

OpenAI, 2023, Assistants API Reference, OpenAI, viewed <viewed-date-here>, https://platform.openai.com/docs/api-reference/assistants.

OpenAI, 2023, Tools for Assistants: File Search, OpenAI, viewed <viewed-date-here>, https://platform.openai.com/docs/assistants/tools/file-search.

REFERENCES

Jiang, Z, Xu, FF, Gao, L, Sun, Z, Liu, Q, Dwivedi-Yu, J, Yang, Y, Callan, J & Neubig, G, 2023. Active Retrieval Augmented Generation. *ArXiv*, abs/2305.06983. Available at: https://api.semanticscholar.org/CorpusID:258615731 <viewed-date-here>

Thakur, A & Vashisth, R, 2024. Loops On Retrieval Augmented Generation (LoRAG). *ArXiv*, abs/2403.15450. Available at: https://api.semanticscholar.org/CorpusID:268680503 <viewed-date-here>

Zhang, X, Xia, M, Couturier, C, Zheng, G, Rajmohan, S & Ruhle, V, 2023. Hybrid Retrieval-Augmented Generation for Real-time Composition Assistance. *ArXiv*, abs/2308.04215. Available at: https://api.semanticscholar.org/CorpusID:260704682 <viewed-date-here>

Zhang, Z, Fang, M & Chen, L, 2024. RetrievalQA: Assessing Adaptive Retrieval-Augmented Generation for Short-form Open-Domain Question Answering. *ArXiv*, abs/2402.16457. Available at: https://api.semanticscholar.org/CorpusID:268033124 <viewed-date-here>.

APPENDIX

APPENDIX A

	method1-mistral_mxbai-embed-large	method2-mistral_mxbai-embed-large	method3-mistral-7B-Instruct-v0.2-GGUF_mxbai-embed-large-v1	method4-gpt-4o_vector-store_file-search	method5-customgpt
Q1	Partial	Fail	Partial	Pass	Pass
Q2	Pass	Partial	Partial	Pass	Pass
Q3	Pass	Fail	Pass	Pass	Pass
Q4	Pass	Pass	Pass	Pass	Pass
Q5	Fail	Pass	Pass	Pass	Pass
Q6	Partial	Partial	Partial	Partial	Partial
Q7	Partial	Pass	Fail	Pass	Pass
Q8	Partial	Pass	Pass	Pass	Pass
Q9	Pass	Pass	Pass	Pass	Pass
Q10	Pass	Partial	Pass	Pass	Partial
Q11	Pass	Fail	Pass	Pass	Pass
Q12	Pass	Fail	Pass	Pass	Pass
Q13	Partial	Partial	Pass	Pass	Partial
Q14	Fail	Partial	Partial	Pass	Pass
Q15	Pass	Pass	Pass	Pass	Pass
Q16	Partial	Partial	Partial	Partial	Partial
Q17	Fail	Pass	Fail	Pass	Pass

Q18	Fail	Pass	Partial	Pass	Pass
Q19	Pass	Fail	Fail	Pass	Partial
Q20	Partial	Fail	Fail	Pass	Pass
Q21	Pass	Pass	Pass	Pass	Pass
Q22	Pass	Pass	Partial	Partial	Pass
Q23	Partial	Fail	Fail	Partial	Pass
Q24	Partial	Fail	Partial	Partial	Partial
Q25	Fail	Fail	Partial	Partial	Partial
Q26	Pass	Pass	Partial	Pass	Pass
Q27	Partial	Pass	Partial	Pass	Pass
Q28	Fail	Fail	Partial	Pass	Pass
Q29	Partial	Partial	Partial	Pass	Pass
Q30	Pass	Pass	Pass	Pass	Pass
Total Pass	13	13	12	24	23
Total Partial	11	7	13	6	7
Total Fail	6	10	5	0	0

Table 1. Response scores given for each question using different RAG implementations.