LLM_LWR_CRAG: Comparative Analysis of Approaches

Luka Nedimović
luka.nedimovic@dmi.uns.ac.rs
Faculty of Sciences, University of Novi Sad, Novi Sad
Faculty of Computing, Union University, Belgrade

Abstract

In this short paper, we aim to conduct a comparative analysis of approaches implemented as a part of for the **LLM_LWR_CRAG** project. We experiment with **12** different chunking, searching and reranking mechanisms, for file retrieval task.

1 Introduction

Retrieval-Augmented Generation (RAG) is an emerging paradigm in natural language processing that combines the strengths of information retrieval and generative models. Unlike traditional language models that rely solely on pre-trained knowledge, RAG systems dynamically retrieve relevant external documents to ground their responses in real-time information. This approach not only improves the factual accuracy of generated content but also enables models to operate effectively in knowledge-intensive tasks. By integrating retrieval with generation, RAG bridges the gap between static knowledge encoding and dynamic information access, offering a promising framework for applications such as question answering, summarization, and dialogue systems.

The evolution of **RAG** traces back to the growing limitations of purely generative language models in handling fact-based or knowledge-intensive queries. Early language models such as **GPT** and **BERT** demonstrated impressive fluency but often struggled with factual accuracy, especially when relying solely on their fixed pre-trained parameters.

To address this, researchers began exploring hybrid approaches that combined retrieval mechanisms with generation. Initial efforts, such as open-domain question answering systems like **DrQA** and **REALM**, introduced the idea of augmenting models with external document retrieval to enhance grounding and factual consistency. This eventually led to the development of **RAG** by **Facebook AI** in 2020, which tightly integrated a retriever and a generator in a single end-to-end architecture.

Since then, **RAG**-like frameworks have rapidly evolved, incorporating dense retrieval, vector databases, and more advanced large language models. These systems are now central to many real-world applications, offering scalable and dynamic solutions for tasks that demand both language understanding and external knowledge access.

2 Task Description

The problem we are aiming to solve is related to standard RAG applications:
Given the **GitHub** repository URL, index the repository and return **Top-K** relevant files (i.e. file paths) for the given query, preferably accompanied by an LLM generated answer.
The metric used for evaluation is **Recall@10**.

3 Experiments

Each experiment comes with a dedicated **YAML** configuration file, that can be simply run by providing its path as the `--config` command-line argument (e.g., `./main.py --config exp_1.yaml`). Experiment results are, by default, stored in `logs/experiments.csv` file.

3.1 Experimental Settings

In the following table reside the general experimental settings, for reproduction purposes:

Component	**Details**
GPU	NVIDIA GeForce RTX 3060 (6GB)
CPU	Intel i7-12650H (16) @ 4.600GHz
RAM	16 GB DDR4
Python Version	Python 3.1.11

Table 1: Experimental Settings

3.2 Experiment Descriptions

We conduct experiments on general, ablative basis. We compare the effects of chunk size and chunk overlap, query augmentation, LLM-generated summaries, hybrid search (using `BM25`) and LLM reranking. In the table below reside the details of each experiment.

Name	**Ch. Size / Overlap**	**Augmentation**	**Metadata**	**BM25**	**Reranker**
exp_1	1200 / 120	N/A	N/A	N/A	N/A
exp_2	1200 / 120	N/A	N/A	N/A	N/A
exp_3	1200 / 120	N/A	Use	N/A	N/A
exp_4	1500 / 150	N/A	Use	N/A	N/A
exp_5	800 / 80	N/A	Use	N/A	N/A
exp_6	1200 / 120	N/A	Use	Use	N/A
exp_7	1200 / 120	N/A	Use	N/A	gpt-4o-mini
exp_8	1200 / 120	N/A	Use	Use	gpt-4o-mini
exp_9	1200 / 120	gpt-4o-mini	Use	Use	gpt-4o-mini
exp_10	1200 / 120	N/A	N/A	Use	ms-marco-MiniLM-L12-v2
exp_11	1200 / 120	N/A	Use	N/A	ms-marco-MiniLM-L12-v2
exp_12	1200 / 120	gpt-4o-mini	Use	N/A	ms-marco-MiniLM-L12-v2

Table 2: Experiment Descriptions

All experiments are conducted using `RecursiveCharacterTextSplitter` for chunking the documents, `ChromaDB` as the vector database, metadata (where "Use") is both LLM summary and code structure, chunk embedding model is set to `text-embedding-3-large` (for maximal efficiency) and $\mathbf{k = 10}$.

3.3 Experiments Results

In the following table, we display the metrics for the aforementioned experiments:

Name	Recall@10
exp_1	73%
exp_2	71%
exp_3	**86%**
exp_4	84%
exp_5	83%
exp_6	75%
exp_7	**86%**
exp_8	78%
exp_9	67%
exp_10	61%
exp_11	81%
exp_12	78%

Table 3: Experiment results

From the results, we can notice that the experiments $\mathbf{3}$ and $\mathbf{7}$ have the highest average **Recall@10** of $\mathbf{86}\%$, with the only difference being the use of the LLM reranker, meaning that the reranker had negligible impact to the quality of the system.

Following them, experiments $\mathbf{4}$, $\mathbf{5}$ and $\mathbf{11}$ have the highest score. Experiments $\mathbf{4}$ and $\mathbf{5}$ use chunking sizes and overlaps of $1500/150$ and $800/80$, being the only two experiments with such deviations. Experiment $\mathbf{11}$ uses a Huggingface reranker: `cross-encoder/ms-marco-MiniLM-L12-v2`, which proved itself to actually generate certain benefit, if metadata is present. The distinction we can see is the lowest scoring system, i.e. experiment $\mathbf{10}$, that does not provide summaries / code structure to the chunk metadata, and uses `BM25` for hybrid search.

Experiment $\mathbf{9}$, which has all of the capabilities enabled (i.e. query augmentation, full metadata generation, `BM25` and a reranker), shows a considerably bad performance, of $\mathbf{67}\%$.

We notice that experiments containing the `BM25`, for hybrid search, generally show poor performance, whilst the cross-encoder reranker leads to generally better evaluations, if not accompanied by metadata.

4 Conclusion

In this short paper, we conduct $\mathbf{12}$ experiments on file retrieval task. Experimental evidence suggests that the cross-encoding rerankers work well with additional metadata (i.e. textual summary of the content), even better than their LLM counterparts. `BM25` for hybrid search, actually, hinders performance. Optimal chunk size is $\mathbf{1200}$, with overlap being $\mathbf{120}$, and addition of textual chunk summary boosts performance considerably. The most optimal architectures achieved make use of metadata and standard LLM embeddings for retrieval.