RAG Report

StudentName:Yueran Ma StudentID:s4931779

1. Introduction

Retrieval-Augmented Generation (RAG)[1], which combines external retrieval with large language models, has come a long way and is considered a state-of-the-art approach toward knowledge-grounded question ansIring. Unlike single-hop QA, which pulls information from a single source, multi-hop requires aggregating evidence across several documents; hence, it increases demands on the retriever and generator's effective partnership. This project systematically compares different firststage retrieval algorithms in a RAG pipeline and considers their subsequent effects on several LLMs, measuring their performance with Recall, Precision, and F1, plus a task-specific threshold metric for additional training. Under limited compute resources, corpus experiments are done as provided in class. The subsequent report describes the methodologies and experimental setup, outlines results and analyses, and reflects on lessons learned and directions in which I hope to take this work.

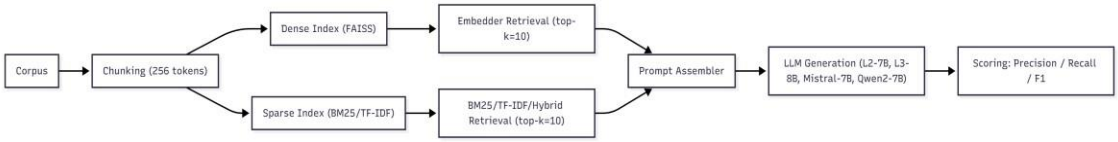
This report assesses four first-stage retrieval algorithms—embedder, BM25, TF-IDF, and hybrid (linear combination of BM25 and TF-IDF). It continues to examine embedding and BM25 retrievers across four LLMs: meta-llama/Llama-2-7b-chat-hf, meta-llama/Meta-Llama-3-8B-Instruct, mistralai/Mistral-7B-Instruct-v0.2, and Qwen/Qwen2-7B-Instruct. This report makes three contributions: (1) a controlled comparison of dense, sparse, and hybrid retrieval under a shared chunking/k regime; (2) a cross-model analysis that maps how retriever choice interacts with different LLMs; and (3) a fully reproducible setup with logged configurations and archived outputs. In brief, lexical methods dominate stage-1 ranking on this corpus, and the strongest end-to-end result is achieved by pairing BM25 with Llama-3-8B; however, the optimal retriever is model-dependent, underscoring that “dense vs. sparse” is not universally decisive. I conclude with a discussion of accuracy-latency-throughput trade-offs and practical guidance for choosing components in resource-bounded RAG. The pipeline does not include a reranking stage; all methods, model configurations, and an experimental setup concerning each retrieval technique are described in detail in the Methodology section.

2. Methodology

This section details the overall RAG pipeline, datasets and preprocessing, first-stage retrieval implementations, LLM generation setup, and the evaluation protocol. I include all information required for reproducibility, while deferring hyperparameter sweeps, full tables, and qualitative case studies to the Experiments section.

Pipeline overview.(**Figure 1**)

I implement a standard RAG workflow: (1) corpus indexing and passage chunking; (2) first-stage retrieval; (3) no reranking stage; (4) LLM generation conditioned on the top-k retrieved passages; and (5) automated metrics plus targeted error analysis. I compare dense (semantic) and sparse (lexical) retrieval to study their different effects on downstream generation quality.



(Figure 1)

Dataset & preprocessing.

All experiments use the course-provided corpus and query set (data/corpus.json, data/rag.json). Documents are split into passages with a chunk size of 256 tokens (via SentenceSplitter), chosen to balance retrieval granularity and context coverage. Text is normalized consistently with the simple tokenizers used by the sparse retrievers. Unless otherwise stated, first-stage retrieval returns top-k = 10 passages per query, and I archive the passage–document mapping for traceable evaluation.

Stage-1 retrieval — implementation details.

Embedder (dense)[2][3] — rankerA.py: Uses HuggingFaceEmbedding through LlamaIndex with the model BAAI/llm-embedder, building a VectorStoreIndex and querying via index.as_retriever(similarity_top_k=10). Reranking is disabled by default (rerank = False).

BM25 (sparse)[4] — rankerB.py: A lightweight in-repo BM25 implementation with k1 = 1.5 and b = 0.75, using the project's simple whitespace tokenizer.

TF-IDF (sparse)[5] — rankerC.py: A TF-IDF vector space with cosine similarity for scoring.

Hybrid (BM25 + TF-IDF) [6][7]— rankerD.py: Normalizes the two score streams and combines them by a linear mixture **s**core = $\alpha \cdot$BM25 + $(1-\alpha) \cdot$TF-IDF with $\alpha = 0.5$ by default; candidate pool and top-k are exposed via arguments (defaults: topk=10). All modules perform the same passage chunking (256 tokens).

Stage-1 evaluation.

I assess first-stage ranking with Hits@4, Hits@10, MAP@10, and MRR, using the provided scripts (e.g., evaluate_stage_rankerA.py and corresponding variants). Gold answers and retrieved texts are normalized (whitespace removal) before overlap checks to reduce formatting artifacts.

LLM generation.

End-to-end RAG is run for two retrievers (embedder and BM25) paired with four instruction-tuned LLMs, yielding eight runs in total:
meta-llama/Llama-2-7b-chat-hf[8] meta-llama/Meta-Llama-3-8B-Instruct[9]
mistralai/Mistral-7B-Instruct-v0.2[10]
Qwen/Qwen2-7B-Instruct[11]
Generation is implemented in the rankerA_rag_*.py and rankerB_rag_*.py scripts via Transformers (AutoModelForCausalLM / AutoTokenizer). I use deterministic decoding (do_sample=False) to minimize stochasticity; max_new_tokens = 32 for regular QA and 8 for comparison-type questions. Prompts concatenate the top-k passages and a fixed instruction: a generic QA prefix (requiring "Insufficient Data" when evidence is missing) or a constrained label set for comparison queries (Yes|No|True|False|Agree|Disagree|Same|Different). Inputs and outputs are recorded to JSON in output/ (e.g., llm-embedder-ranker.json → llama3-8b-rankerA.json), enabling exact reproduction.

Evaluation protocol & reproducibility.

RAG predictions are evaluated with Precision, Recall, and F1 using token/wordoverlap scoring (evaluate_rag_rankerA.py, evaluate_rag_rankerB.py). For

comparison and null queries, answers are normalized to canonical labels or "Insufficient Data" before scoring. I fix decoding to be deterministic, log hardware configuration, and archive all intermediate files (stage-1 rankings and final generations). Code, file names, and default settings are visible in the repository modules cited above.

3. Experiments

This section specifies research questions, the experimental matrix, evaluation protocol, decoding strategy, and procedure. I then report an issue log driven by my actual process, including a local→cloud migration and Git synchronization pipeline.

3.1 Research questions

Stage-1 effectiveness: Among four first-stage retrievers (embedder, BM25, TF-IDF, Hybrid BM25+TF-IDF), which achieves the best retrieval quality on the multi-hop dataset measured by Hits@k (k∈{4,10}), MRR@10, MAP@10?

Retriever→Generator: Holding the LLM fixed, how does dense vs. sparse retrieval (embedder vs. BM25) change end-to-end Precision/Recall/F1? Are the effects consistent across models or do I observe cross-stage interactions?

Model choice under identical context: Holding the retriever fixed (embedder; BM25), which of the four LLMs (Llama-2-7B-chat, Llama-3-8B-Instruct, Mistral-7BInstruct-v0.2, Qwen2-7B-Instruct) yields the highest F1?

3.2 Experimental matrix

Eight end-to-end systems: two retrievers (embedder, BM25) × four LLMs (Llama-27B-chat, Llama-3-8B-Instruct, Mistral-7B-Instruct-v0.2, Qwen2-7B-Instruct). Stage-1 also reports TF-IDF and Hybrid (BM25+TF-IDF).

3.3 Evaluation protocol

Stage-1: Hits@4/10, MRR, MAP@10 on the course queries; chunks of 256 tokens; default top-k=10.

End-to-end: Precision/Recall/F1 with canonicalization of comparison/null queries to fixed labels or "Insufficient Data". All runs log configs and write JSON to output.

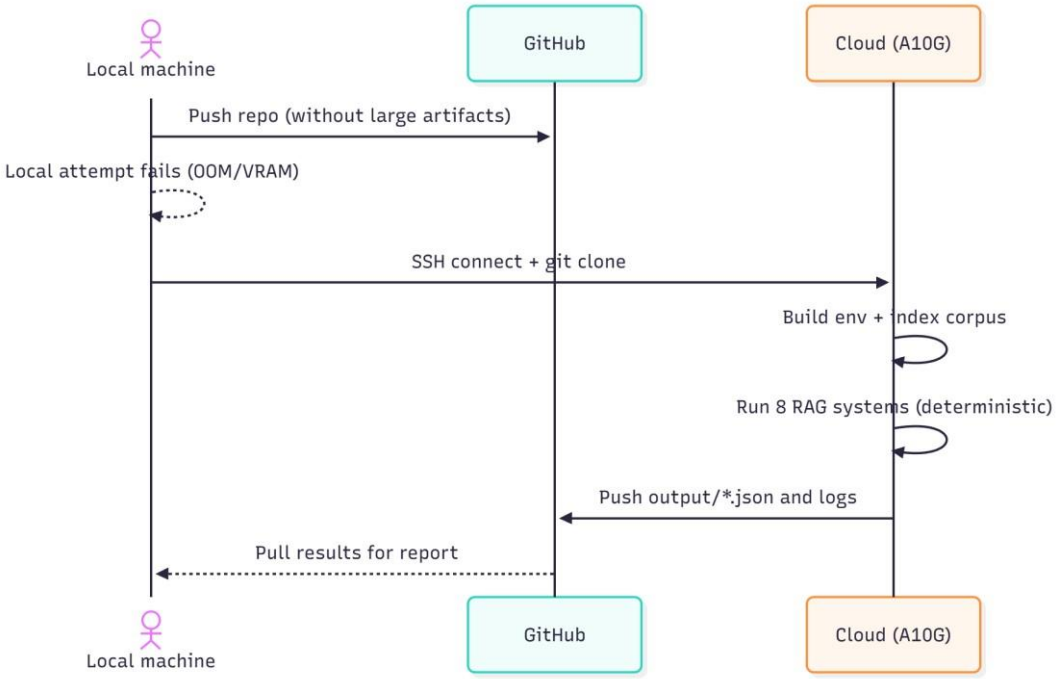
3.4 Procedure

Index dense/sparse → retrieve top-k=10 → assemble prompt → generate deterministically → score and archive → stage-1 ablations on k and α.

3.5 Infrastructure & workflow(Figure 2)

I initially attempted to run locally but 7B-class models exceeded local GPU/VRAM and CPU RAM limits. I migrated to a cloud A10G instance and executed all experiments there. At the beginning, uploading the full corpus via web UI failed; I

switched to Git to ensure a complete and consistent dataset. After each run, outputs were versioned and pushed back to GitHub for traceability.



(Figure 2)

4. Conclusions

Based on eight end-to-end RAG systems (two retrievers × four LLMs) and four stage1 retrievers, I draw three main conclusions:

(1) Stage-1 lexical methods were strongest on retrieval quality. BM25, TF-IDF, and the Hybrid (BM25+TF-IDF, α=0.5) achieved higher Hits@10 and MRR than the dense embedder on this corpus (see Stage-1 plots: Hits@10 and MRR; Figures 3-6). This indicates strong lexical signals in the dataset at k=10. Figure 3:

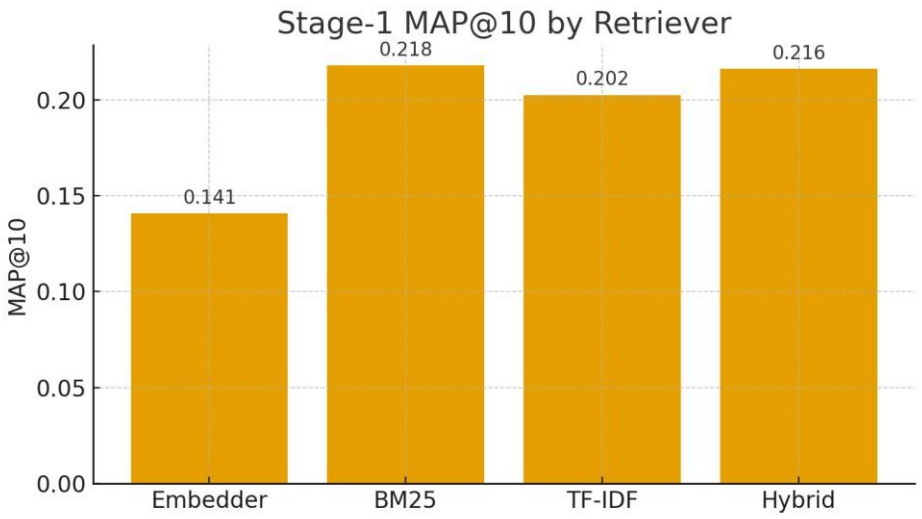


Figure 4:

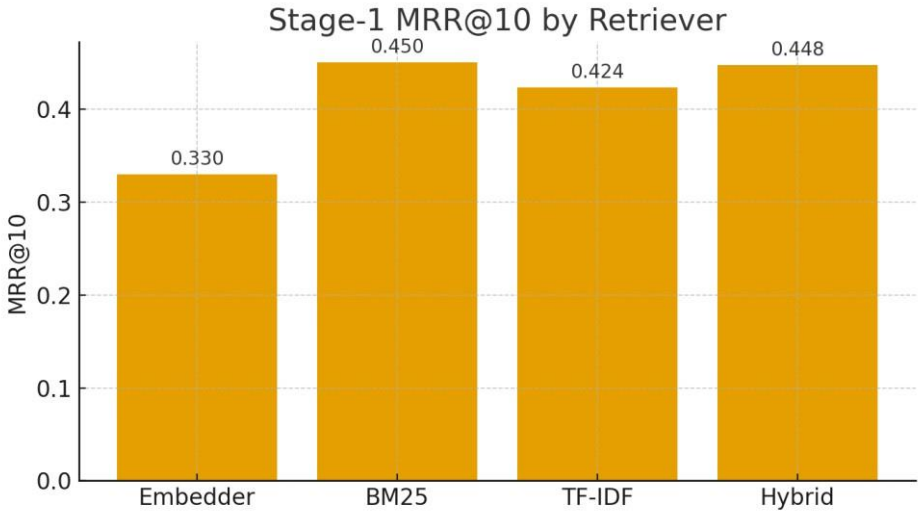


Figure 5:

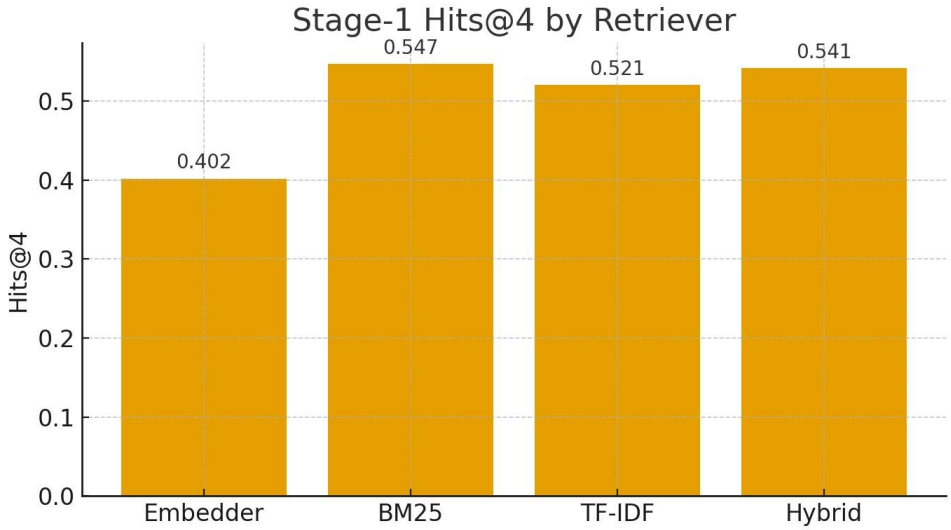
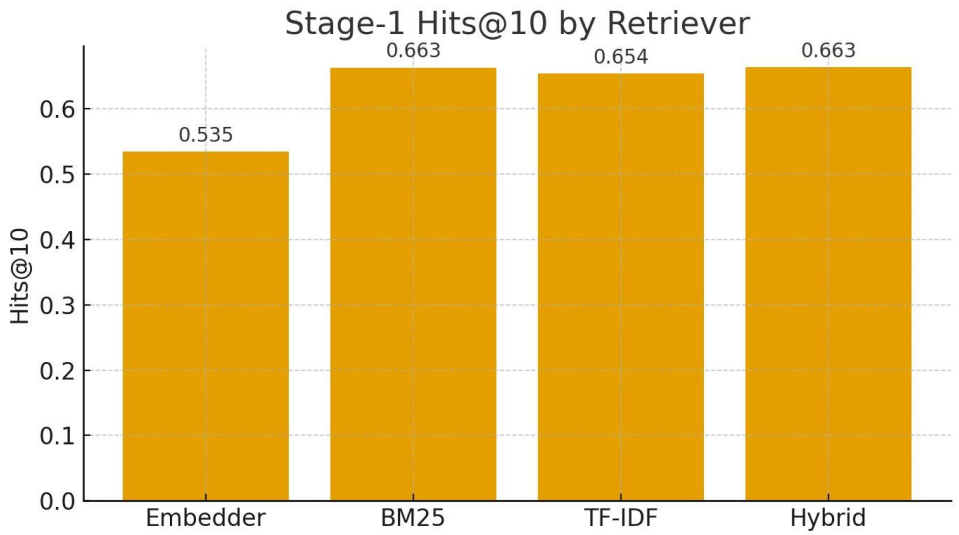
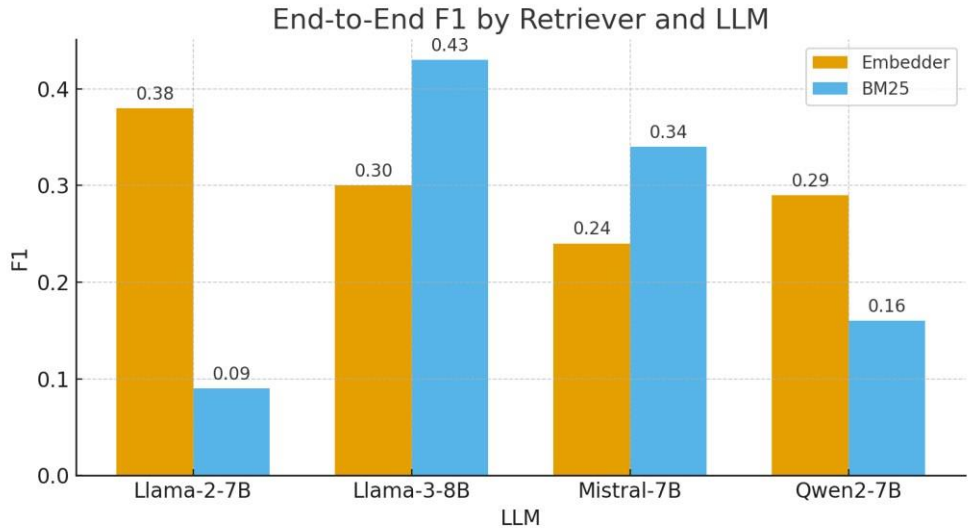


Figure 6:



(2) Best end-to-end system: Llama-3-8B with BM25. Among the eight RAG runs, the highest F1 was achieved by BM25 + Llama-3-8B. However, the interaction is model-dependent: Llama-2-7B paired better with the embedder, while Mistral-7B and Llama-3-8B benefited more from BM25 (see end-to-end F1 bar chart, Figure 7).

Figure 7:



(3) When using Llama-2-7B, a dense embedder can be preferable despite weaker stage-1 recall, underscoring that retriever choice should be LLM-specific rather than universally “dense vs. sparse”. With a fixed budget at k=10, the Hybrid retriever tracked BM25 closely; gains from mixing BM25 and TF-IDF were modest on this corpus.

Limitations.

This study omits a reranking stage and is run under constrained compute on a single corpus with fixed passage chunking (256 tokens) and a default retrieval depth of top-k=10. These choices favor reproducibility but may under-estimate the headroom available with rerankers or larger contexts. Beyond these scope constraints, three further limitations remain:

(1) Although the report documents that reranking is disabled, I did not evaluate any reranking baselines. This likely underestimates the true potential of several first-stage retrievers. Future work: add at least one cross-encoder reranker and compare its impact on Hits@k and end-to-end F1, including latency/throughput overhead.

(2) Several settings were left at defaults, notably the hybrid mixture coefficient (α=0.5), retrieval depth (top-k=10), and passage chunk size (256). I also did not systematically vary BM25 parameters (k1, b) or the decoding budget. Future work: run targeted sweeps (e.g., α∈[0.2,0.8], top-k∈{5,10,20}, chunk size∈{128,256,384}, BM25 k1/b grids) and report sensitivity curves to quantify robustness.

5. Acknowledgement:

I would first like to express my sincere gratitude to my supervisor, Professor Shane, for the successful completion of this paper. Throughout the research process, Professor Shane provided clear guidance and tremendous help by patiently answering my many questions through announcements and timely email communication, which enabled the research to proceed smoothly.
In addition, this study also utilized Generative AI (GenAI) tools at multiple stages. I primarily used GenAI for rapid literature searches, translation and summarization of foreign literature, and to gain a deeper understanding of key algorithm principles. Moreover, during the experimental deployment phase, GenAI provided me with the necessary shell commands to upload the project from a GitHub repository to a cloud GPU server, which significantly enhanced my research efficiency. I hereby also express my gratitude for the support provided by this technology.

6. Reference:

[1] P. Lewis *et al.*, 'Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks', Apr. 12, 2021, *arXiv*: arXiv:2005.11401. doi: 10.48550/arXiv.2005.11401.

[2] V. Karpukhin *et al.*, 'Dense Passage Retrieval for Open-Domain Question Answering', Sept. 30, 2020, *arXiv*: arXiv:2004.04906. doi: 10.48550/arXiv.2004.04906.

[3] N. Reimers and I. Gurevych, 'Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks', Aug. 27, 2019, *arXiv*: arXiv:1908.10084. doi: 10.48550/arXiv.1908.10084.

[4] S. Robertson and H. Zaragoza, 'The Probabilistic Relevance Framework: BM25 and Beyond', *Found. Trends® Inf. Retr.*, vol. 3, no. 4, pp. 333–389, 2009, doi: 10.1561/1500000019.

[5] G. Salton and C. Buckley, ‘Term-weighting approaches in automatic text retrieval’, *Inf. Process. Manag.*, vol. 24, no. 5, pp. 513–523, Jan. 1988, doi: 10.1016/03064573(88)90021-0.

[6] J. H. Lee, ‘Analyses of multiple evidence combination’, in *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’97*, Philadelphia, Pennsylvania, United States: ACM Press, 1997, pp. 267–276. doi: 10.1145/258525.258587.

[7] D. K. Harman, *Overview of the Third Text REtrieval Conference (TREC-3)*. DIANE Publishing, 1995.

[8] H. Touvron *et al.*, ‘Llama 2: Open Foundation and Fine-Tuned Chat Models’, July 19, 2023, *arXiv*: arXiv:2307.09288. doi: 10.48550/arXiv.2307.09288.

[9] A. Grattafiori *et al.*, ‘The Llama 3 Herd of Models’, Nov. 23, 2024, *arXiv*: arXiv:2407.21783. doi: 10.48550/arXiv.2407.21783.

[10] A. Q. Jiang *et al.*, ‘Mistral 7B’, Oct. 10, 2023, *arXiv*: arXiv:2310.06825. doi: 10.48550/arXiv.2310.06825.

[11] A. Yang *et al.*, ‘Qwen2 Technical Report’, Sept. 10, 2024, *arXiv*: arXiv:2407.10671. doi: 10.48550/arXiv.2407.10671.