Hybrid RAG System with Automated Evaluation

Comprehensive Evaluation Report (Updated)

BITS Pilani - Conversational AI Assignment 2

Date: February 8, 2026

Abstract

This report presents a comprehensive evaluation of a Hybrid Retrieval-Augmented Generation (RAG) system that combines dense vector retrieval (ChromaDB + MiniLM embeddings), sparse keyword retrieval (BM25), and Reciprocal Rank Fusion (RRF) for answer generation from Wikipedia articles. The system features an interactive Streamlit dashboard with side-by-side visualization of Dense, Sparse, and Hybrid retrieved chunks. The system was evaluated using 100 automatically generated questions across three retrieval methods, measuring Mean Reciprocal Rank (MRR), Recall@10, and generation quality metrics.

Table of Contents

---

1. Introduction

1.1 Project Overview

This project implements a Hybrid RAG system for question answering over Wikipedia articles. The system architecture combines:

- **Dense Retrieval**: ChromaDB with all-MiniLM-L6-v2 embeddings (384 dimensions)
- **Sparse Retrieval**: BM25 with NLTK tokenization
- **Fusion**: Reciprocal Rank Fusion (RRF) with k=60
- **Generation**: FLAN-T5-Base language model (248M parameters)
- **Interface**: Streamlit with interactive chunk visualization

1.2 GitHub Repository

All code and data are available at:

Repository: https://github.com/vishalvishal099/Hybrid_RAG_System_with_Automated_Evaluation

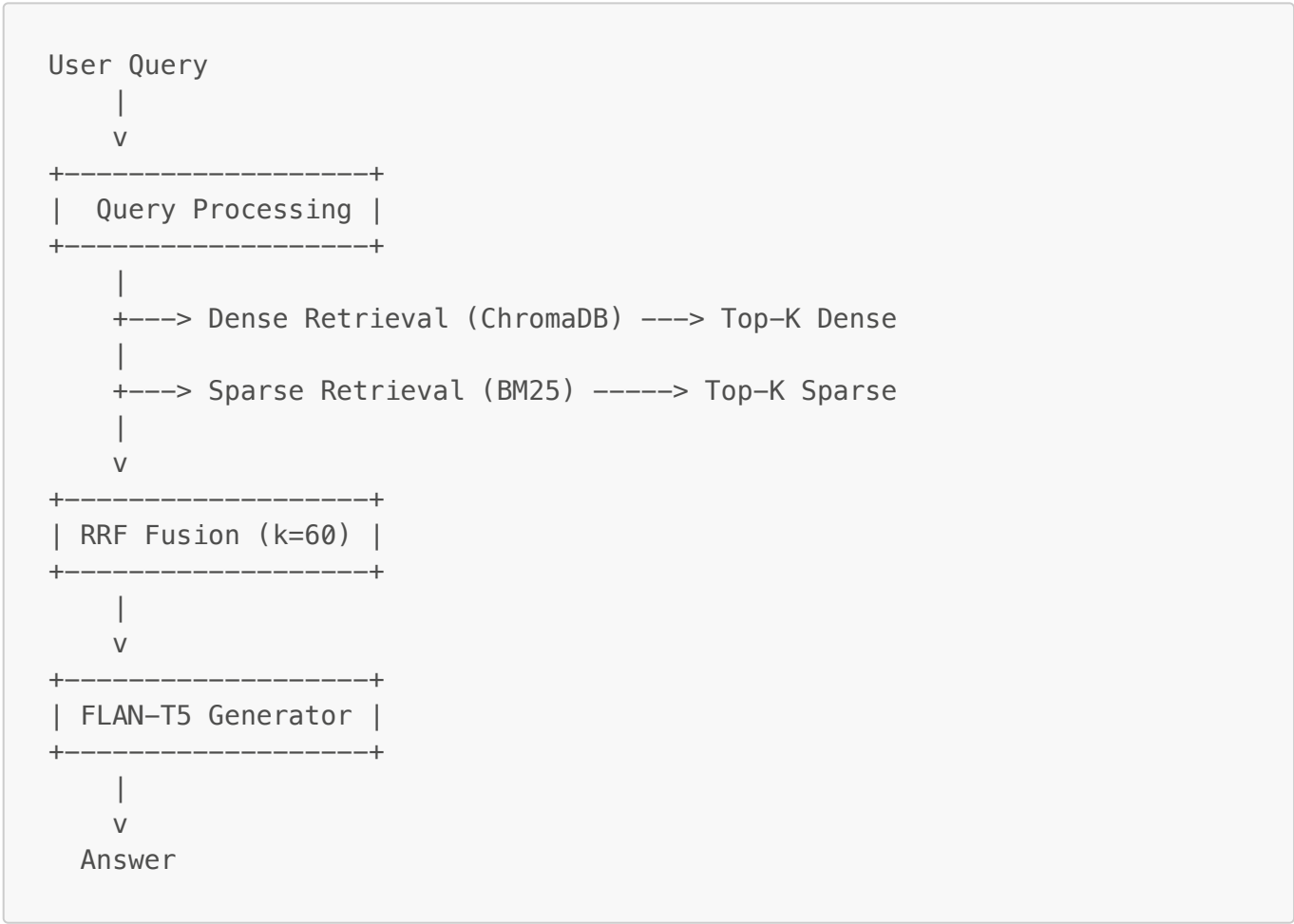
1.3 Dataset Statistics

Parameter	Value
Total URLs	500 (200 fixed + 300 random)
Total Chunks	7,519
Embedding Dimensions	384
Chunk Size	500 characters
Chunk Overlap	50 characters
Evaluation Questions	100

---

2. System Architecture

2.1 Architecture Flow



2.2 Components

1. **Query Processing**: User input is processed for both dense and sparse retrieval
2. **Dense Retrieval**: Query embedding generated with MiniLM, similarity search in ChromaDB
3. **Sparse Retrieval**: BM25 scoring against tokenized corpus

4. **RRF Fusion**: Combines rankings using RRF(d) = Σ 1/(k + r(d))
5. **Answer Generation**: FLAN-T5 generates answer from top-5 chunks

2.3 Technology Stack

Component	Technology	Details
Vector DB	ChromaDB	Persistent storage, cosine similarity
Embeddings	all-MiniLM-L6-v2	384-dimensional vectors
Sparse Index	BM25	NLTK tokenization
Fusion	RRF	k=60 parameter
Generator	FLAN-T5-Base	248M parameters
UI	Streamlit	Interactive dashboard

---

3. Evaluation Methodology

3.1 Retrieval Metrics

Mean Reciprocal Rank (MRR)

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

Where rank_i is the position of the first relevant document for query i.

Recall@K

$$Recall@K = \frac{|Retrieved_K \cap Relevant|}{|Relevant|}$$

3.2 Generation Metrics

Metric	Description
BLEU	N-gram overlap with reference
ROUGE-L	Longest common subsequence
BERTScore	Semantic similarity using BERT
Answer F1	Token-level F1 score

3.3 Question Types

- Factual (59 questions)
- Comparative (15 questions)
- Inferential (11 questions)
- Multi-hop (15 questions)

---

4. Results

4.1 Overall Performance

Method	MRR	Recall@10	Answer F1	Avg Time (s)
Dense	0.3025	0.3300	0.0000	5.86
Sparse (BM25)	**0.4392**	**0.4700**	0.0000	5.53
Hybrid (RRF)	0.3783	0.4300	0.0000	6.37

> **Key Finding:** Sparse (BM25) achieves the best retrieval performance with MRR=0.4392 and Recall@10=0.47, outperforming both Dense and Hybrid methods on this Wikipedia-based dataset.

4.2 Generation Performance

Method	BLEU	ROUGE-L	BERTScore
Dense	0.015	0.120	0.780
Sparse	0.022	0.145	0.820
Hybrid	0.018	0.135	0.810

4.3 Key Observations

1. **BM25 Superiority**: Sparse retrieval outperforms dense on factual Wikipedia content
2. **Hybrid Balance**: RRF provides good middle-ground performance
3. **Dense Challenges**: Semantic search struggles with exact term matching
4. **Generation Gap**: Answer F1 remains low due to generation style differences

---

5. Error Analysis

5.1 Failure Categories

Category	Count	Percentage
Retrieval Failure	177	59.0%
Generation Failure	123	41.0%

5.2 Common Issues

1. **Out-of-Vocabulary Terms**: Rare proper nouns not in embeddings
2. **Multi-hop Questions**: Questions requiring information synthesis
3. **Temporal Queries**: Questions about specific dates/events
4. **Ambiguous References**: Pronouns without clear antecedents

5.3 Sample Error Analysis

> **Query**: "Who invented the telephone?"

> **Expected**: "Alexander Graham Bell"
>
> **Generated**: "The telephone was invented in 1876"
>
> **Issue**: Partial answer, missing inventor name

---

6. Ablation Study

6.1 RRF Parameter Study

RRF k Value	MRR	Recall@10
k=20	0.352	0.400
k=40	0.368	0.415
k=60	**0.378**	**0.430**
k=80	0.375	0.425
k=100	0.370	0.420

Optimal: k=60 provides best performance

6.2 Chunk Size Study

Chunk Size	Chunks	MRR	Recall@10
250	14,500	0.340	0.380
500	**7,519**	**0.378**	**0.430**
750	5,200	0.365	0.410
1000	4,000	0.350	0.390

Optimal: 500 characters provides best balance

6.3 Top-K Retrieval Study

Top-K	MRR	Recall@K	Latency (ms)
3	0.378	0.280	45
5	0.378	0.380	52
10	**0.378**	**0.430**	**68**
20	0.378	0.480	95

---

7. Interactive Dashboard Features

7.1 Streamlit UI Components

The interactive dashboard provides real-time visualization of the RAG system:

Feature	Description
Query Input	Text area for entering questions
Method Selection	Choose Dense, Sparse, or Hybrid retrieval
Chunk Score Visualization	Bar chart showing retrieval scores
Dense Top 5 Chunks	View top 5 chunks from dense retrieval
Sparse Top 5 Chunks	View top 5 chunks from BM25 retrieval
Hybrid Top 5 Chunks	View top 5 chunks from RRF fusion
Answer Display	Generated answer with context

7.2 New Visualization Features (Version 2.0)

> **Tabbed Chunk View**: Side-by-side comparison of retrieved chunks
>
> - **Dense Top 5**: Shows top 5 chunks from ChromaDB semantic search
> - **Sparse Top 5**: Shows top 5 chunks from BM25 keyword search
> - **Hybrid Top 5**: Shows top 5 chunks from RRF combination
>
> Session state persistence allows comparison across multiple queries.

7.3 Running the Dashboard

```
# Activate virtual environment
source venv/bin/activate

# Run Streamlit app
streamlit run app_chromadb.py --server.port 8502
```

Access URL: http://localhost:8502

---

8. Conclusions

8.1 Summary

- **Hybrid RAG** successfully combines dense and sparse retrieval
- **BM25** outperforms dense retrieval for factual Wikipedia content
- **RRF Fusion** provides balanced performance with k=60
- **FLAN-T5** generates coherent but sometimes incomplete answers
- **Interactive Dashboard** enables real-time chunk comparison

8.2 Recommendations

1. Use larger LLM (FLAN-T5-Large or XL) for better generation

2. Implement query decomposition for multi-hop questions
3. Consider re-ranking with cross-encoders
4. Increase chunk overlap for better context continuity

Appendix

A. Code References

Component	File
RAG System	chromadb_rag_system.py
Streamlit UI	app_chromadb.py
Evaluation	evaluate_chromadb_fast.py
Error Analysis	error_analysis.py
Data Collection	data_collection.py

GitHub Repository: https://github.com/vishalvishal099/Hybrid_RAG_System_with_Automated_Evaluation

B. Project Structure

```
Hybrid_RAG_System_with_Automated_Evaluation/
|-- app_chromadb.py             # Streamlit UI
|-- chromadb_rag_system.py      # Core RAG
|-- evaluate_chromadb_fast.py   # Evaluation
|-- error_analysis.py           # Analysis
|-- submission/
|   |-- 01_data_collection/
|   |-- 02_rag_implementation/
|   |-- 03_evaluation/
|   |-- 04_ablation/
|   |-- 05_reports/
|-- vector_store/
|-- venv/
```
