Assignment 2 – Hybrid RAG System with Automated Evaluation

Important Dates

- Roll-out Date: **15 January 2026**
- Doubt Clarification Deadline: **22 January 2026**
- Submission Deadline: **8 February 2026**

Please Note: All clarifications must be raised strictly within the first 7 days from the assignment roll-out, i.e., by 22 January 2026. Any questions submitted after this date will not be entertained.

Objective

Build a Hybrid Retrieval-Augmented Generation (RAG) system combining dense vector retrieval, sparse keyword retrieval (BM25), and Reciprocal Rank Fusion (RRF) to answer questions from 500 Wikipedia articles. Evaluate using an automated framework with 100 generated questions.

Dataset Requirements

Wikipedia URL Collection (500 URLs per group):

Fixed Set (200 URLs): Each group must first sample a unique set of 200 Wikipedia URLs (minimum 200 words per page) covering diverse topics. Store these in a JSON file (fixed_urls.json). **No two groups should share the same 200 URLs.** These URLs remain constant across all indexing operations.

Random Set (300 URLs): For each indexing run, randomly sample 300 additional Wikipedia URLs (minimum 200 words per page). These should change every time the system is rebuilt/indexed.

Total Corpus: 200 fixed + 300 random = 500 URLs. Extract, clean, and chunk the text (200-400 tokens with 50-token overlap). Store with metadata (URL, title, unique chunk IDs).

Part 1: Hybrid RAG System (10 Marks)

1.1 Dense Vector Retrieval

Use a sentence embedding model (e.g., all-MiniLM-L6-v2, all-mpnet-base-v2) to embed chunks. Build a vector index (FAISS/ChromaDB) and retrieve top-K chunks via cosine similarity.

1.2 Sparse Keyword Retrieval

Implement BM25 algorithm for keyword-based retrieval. Build index over chunks and retrieve top-K results.

1.3 Reciprocal Rank Fusion (RRF)

For each query, retrieve top-K chunks from both dense and sparse methods. Combine using RRF: RRF_score(d) = Σ 1/(k + rank_i(d)) where k=60. Select top-N chunks by RRF score for final context.

1.4 Response Generation

Use an open-source LLM (e.g., DistilGPT2, Flan-T5-base, Llama-2-7B). Concatenate top-N chunks with query and generate answers within context limits.

1.5 User Interface

Build with Streamlit/Gradio/Flask. Display: user query input, generated answer, top retrieved chunks with sources, dense/sparse/RRF scores, and response time.

Part 2: Automated Evaluation (6 + 4 Marks)

2.1 Question Generation (Automated)

Generate 100 Q&A pairs from Wikipedia corpus using LLMs or extraction methods. Include diverse question types: factual, comparative, inferential, multi-hop. Store with ground truth, source IDs, and question categories.

2.2 Evaluation Metrics

2.2.1 Mandatory Metric (Basic - 2 Marks)

Mean Reciprocal Rank (MRR) - URL Level: Calculate MRR at the URL level (not chunk level). For each question, find the rank position of the first correct Wikipedia URL in the retrieved results. MRR = average of 1/rank across all questions. This measures how quickly the system identifies the correct source document.

2.2.2 Additional Custom Metrics (4 Marks)

Select and implement 2 additional metrics beyond the mandatory MRR. For each metric, you must:

1. Justify Selection: Explain why this metric is important for evaluating your RAG system.
2. Calculation Method: Provide detailed mathematical formulation and implementation details.
3. Interpretation: Explain what the scores indicate about system performance and how to interpret results.

Suggested metric categories: Answer quality (EM, F1, BLEU, ROUGE, BERTScore, Semantic Similarity), Retrieval quality (Precision@K, Recall@K, NDCG@K, Hit Rate), Context relevance (Contextual Precision, Contextual Recall), Faithfulness (hallucination detection, answer grounding), Efficiency (response time, latency breakdown), or Novel custom metrics designed by your team.

2.3 Innovative Evaluation (4 Marks)

Demonstrate creativity through advanced techniques such as:

- Adversarial Testing: Challenging questions (ambiguous, negated, multi-hop), paraphrasing robustness, unanswerable questions for hallucination detection.
- Ablation Studies: Compare dense-only, sparse-only, and hybrid performance. Experiment with different K, N, and RRF k values.
- Error Analysis: Categorize failures (retrieval, generation, context issues) by question type with visualizations.
- LLM-as-Judge: Use LLM to evaluate factual accuracy, completeness, relevance, and coherence with automated explanations.
- Confidence Calibration: Estimate answer confidence and measure correlation with correctness using calibration curves.
- Novel Metrics: Custom metrics for entity coverage, answer diversity, hallucination rate, or temporal consistency.
- Interactive Dashboard: Real-time metrics, question breakdowns, retrieval visualizations, and method comparisons.

2.4 Automated Pipeline

Build a single-command pipeline that loads questions, runs a RAG system, computes all metrics, and generates comprehensive reports (PDF/HTML) with structured output (CSV/JSON).

2.5 Evaluation Report Contents

Overall performance summary with MRR and custom metrics averages. Detailed justification for the 2 selected custom metrics including: why chosen, calculation methodology, and interpretation guidelines. Results table (Question ID, Question, Ground Truth, Generated Answer, MRR, Custom Metric 1, Custom Metric 2, Time). Visualizations: metric comparisons, score distributions, retrieval heatmaps, response times, ablation results. Error analysis with failure examples and patterns.

Submission Requirements

Submit one ZIP file per group: Group_<Number>_Hybrid_RAG.zip

Required Contents:

1. Code (.ipynb or .py): Complete RAG implementation (data collection, preprocessing, dense/sparse retrieval, RRF, generation) with detailed comments and markdown.
2. Evaluation: Question generation script, 100-question dataset (JSON/CSV), evaluation pipeline, metrics implementation, and innovative components.
3. Report (PDF): Architecture diagram, evaluation results with tables/visualizations, innovative approach description, ablation studies, error analysis, and 3+ system screenshots.
4. Interface: Hosted app link (Streamlit/Gradio/HF Spaces) OR standalone app with setup instructions.
5. README.md: Installation steps, dependencies, run instructions (system + evaluation), and fixed 200 Wikipedia URLs list in JSON format.
6. Data: fixed_urls.json (200 URLs), preprocessed corpus, vector database, 100-question dataset, and evaluation results (all files or regeneration instructions).

Technical Notes

Required Libraries: sentence-transformers, faiss-cpu, rank-bm25, transformers, wikipedia-api, beautifulsoup4, nltk, rouge-score, bert-score, scikit-learn.

Resources: Google Colab, Kaggle Notebooks, or campus GPU clusters. Use only open-source models and tools.

Key to Success: Innovation in evaluation is critical for full marks. Think beyond standard metrics, be creative!

Good luck!