CS6913 Web Search Engines Homework 3

Dong Li
dl5214@nyu.edu

Nov 2024

Abstract

This report presents a comparative analysis of three retrieval systems implemented for information retrieval tasks using the MS MARCO Passage Ranking dataset. System 1 uses the BM25 scoring function for exact term matching, System 2 employs a graph-based HNSW algorithm for semantic embedding retrieval, and System 3 integrates BM25 retrieval with embedding-based re-ranking. The evaluation focuses on retrieval performance (MAP, MRR, Recall@100, and NDCG), computational efficiency, and memory usage. Results highlight the trade-offs between efficiency and semantic relevance among the three systems, with System 3 achieving a balanced performance by leveraging the strengths of both BM25 and embeddings.

Keywords: BM25, HNSW, Dense Vector Retrieval, Information Retrieval

1 Introduction

Efficient retrieval of relevant information from large datasets is a core challenge in information retrieval. Traditional methods like BM25 are computationally efficient but limited in capturing semantic relevance. Embedding-based approaches address this limitation by modeling semantic relationships through dense vector representations but require higher computational and memory resources.

This assignment reconstructs a BM25-based system with an inverted index on a subset of the MS MARCO Passage Ranking dataset and introduces two additional systems: a graph-based embedding retrieval system using the Hierarchical Navigable Small World (HNSW) algorithm and a hybrid system combining BM25 with embedding-based re-ranking. These systems are evaluated for retrieval quality and efficiency, highlighting their strengths and trade-offs.

The structure of this report is as follows: Section 2 discusses the principles and workflows of the three retrieval systems. Section 3 describes the dataset, evaluation metrics, and benchmarks used to assess system performance. Section 4 presents the results and analysis, including performance comparisons and the impact of parameter configurations. Section 5 examines the trade-offs between computational efficiency and retrieval quality, highlights the strengths and limitations of each system. Finally, Section 6 summarizes the findings and contributions of this work.

2 Principles and Workflows of Query Systems

This chapter provides an overview of the three retrieval systems implemented in this assignment, the dataset used, and the detailed workflows of each system. The systems are designed to explore different methods of information retrieval, ranging from traditional BM25-based approaches to embedding-based re-ranking.
Systems Overview: - **System 1:** BM25 retrieval in disjunctive mode, using word-based matching. - **System 2:** Graph-based retrieval using the Hierarchical Navigable Small World (HNSW) algorithm. - **System 3:** A hybrid system combining BM25 for initial ranking with embedding-based re-ranking.

The graph-based HNSW system is implemented using the Python `faiss` library, while System 1's BM25 operations are implemented in C++. Python serves as the primary language for overall orchestration.

The evaluation benchmarks include:

- **Mean Average Precision (MAP):** Measures precision across multiple levels of recall.
- **Normalized Discounted Cumulative Gain (NDCG@10, NDCG@100):** Evaluates ranking quality based on relevance.
- **Mean Reciprocal Rank (MRR):** Focuses on the rank position of the first relevant result.
- **Recall@100:** Computes the proportion of relevant documents retrieved in the top 100 results.

2.1 Dataset

The dataset used is the **MS MARCO Passage Ranking** dataset, which consists of 8.8 million passages curated for passage retrieval tasks. For this assignment, a subset of 1 million passages is provided with embeddings, pre-selected from the original dataset. The dataset is organized as follows:

- `msmarco_passage_embeddings_subset.h5`: Contains the IDs and dense embeddings of the 1 million selected passages.
- `msmarco_dev_eval_embeddings.h5`: Contains the IDs and dense embeddings of all queries used for evaluation.
- `msmarco_passages_subset.tsv`: Allows inspection of passage IDs included in the 1 million selected passages, along with their textual content.
- `queries.eval.tsv` and `queries.dev.tsv`: Provide textual queries for evaluation and development purposes, respectively.

This dataset subset reduces computational overhead while retaining a representative sample for meaningful evaluation. Embeddings for passages and queries are precomputed using a transformer-based model.

2.2 System 1: BM25 Query System

System 1 is based on the BM25 ranking function, a widely used probabilistic retrieval model. It scores documents based on their term frequency (TF), inverse document frequency (IDF), and query-document length normalization.

Key Features:

- **INDEX_SUBSET and RETRIEVE_CONTENT Flags:**
  - `INDEX_SUBSET = 0`: Indexes all 8.8 million passages in the full MS MARCO dataset.
  - `INDEX_SUBSET = 1`: Indexes only the 1 million subset passages. Passages are included if their `docID` exists in `msmarco_passages_subset.tsv`.
  - `RETRIEVE_CONTENT`: Determines whether passage content is retrieved alongside `docID` and BM25 scores.
- **NUM_TOP_RESULT:** Configured to retrieve the top 500 results for each query, ensuring sufficient candidate passages for downstream re-ranking in System 3.

Workflow: For a given query ID, the corresponding text is retrieved from `queries.eval.tsv` or `queries.dev.tsv`. The BM25 system processes the query to retrieve and rank passages based on BM25 scores. The top results are returned for evaluation or further processing.

2.3 System 2: HNSW Graph-Based System

2.3.1 Introduction to HNSW

The **Hierarchical Navigable Small World (HNSW)** algorithm [1] is a graph-based approximate nearest neighbor (ANN) search method. It organizes data points into multiple layers of graphs, enabling efficient navigation and retrieval of nearest neighbors.

Key Parameters:

- `m`: Maximum number of connections per node. Larger `m` improves recall but increases index size and construction time.
- `ef_construct`: Candidate list size during index construction. Higher values enhance accuracy but slow down the building process.
- `ef_search`: Candidate list size during querying. Larger values improve search accuracy but increase query latency.

2.3.2 Workflow

The HNSW workflow involves two main phases:

- **Index Building:** Using `faiss`, passage embeddings are normalized for cosine similarity search. The graph-based index is constructed with `faiss.IndexHNSWFlat(embedding_dim, m)`. The parameters `m` and `ef_construct` are tuned to balance accuracy and efficiency.
- **Query Processing:** For a given query, its embedding is retrieved and searched against the index. The top-K ANN results are ranked by cosine similarity.

2.4 System 3: Mixed BM25 and Embedding Re-ranking System

System 3 integrates the advantages of BM25 and embedding-based retrieval to enhance ranking performance. It first retrieves passages using BM25 and then re-ranks them based on semantic similarity.

Workflow:

- **Step 1:** Retrieve the top 500 passages from System 1 using BM25 scores.
- **Step 2:** Extract embeddings for the retrieved passages from the precomputed dataset.
- **Step 3:** Compute cosine similarity between the query embedding and passage embeddings.
- **Step 4:** Re-rank passages based on similarity scores and return the top 100 as the final output.

Modes of Operation:

- `QUERY_BM25 = true`: Queries System 1 dynamically to retrieve BM25 results.
- `QUERY_BM25 = false`: Reads precomputed BM25 output files to skip query processing in System 1, improving efficiency.

By combining BM25 and embeddings, System 3 captures semantic similarities missed by word-level matching. However, its reliance on the initial BM25 retrieval means it cannot resolve cases where relevant documents are excluded from the initial results due to term mismatches or morphological variations.

3 Evaluation Benchmarks

3.1 Explanation of Evaluation Metrics

To assess the effectiveness of search systems, we employ the following evaluation metrics:

Mean Average Precision (MAP) MAP evaluates the precision of a system by considering the ranked position of all relevant documents for a query [2]. It computes the average precision for each query and then averages it across all queries. This metric is suited for datasets with both binary (0/1) and multiple relevance labels (e.g., 0, 1, 2, 3), making it versatile for evaluating retrieval systems. The formula for MAP is as follows:

$$\text{AP} = \frac{1}{R} \sum_{k=1}^{n} \text{Precision}(k) \cdot \text{rel}(k), \tag{1}$$

where R is the total number of relevant documents, n is the total number of retrieved documents, $\text{Precision}(k)$ is the precision at rank k, and $\text{rel}(k)$ is a binary indicator (1 if the document at rank k is relevant, 0 otherwise). MAP is then calculated as:

$$\text{MAP} = \frac{1}{Q} \sum_{q=1}^{Q} \text{AP}(q), \tag{2}$$

where Q is the total number of queries.

Mean Reciprocal Rank (MRR) MRR focuses on the rank of the first relevant document for a query [3]. It calculates the reciprocal of the rank of the first relevant document and averages this value over all queries. This metric is particularly useful for binary relevance datasets where identifying the most relevant document is critical. The formula for MRR is given by:

$$\text{MRR} = \frac{1}{Q} \sum_{q=1}^{Q} \frac{1}{\text{rank}_q}, \tag{3}$$

where rank_q is the rank position of the first relevant document for query q.

Recall@100 Recall measures the proportion of all relevant documents retrieved by the system out of the total relevant documents in the dataset [4]. Recall@100 indicates the recall value when considering the top 100 results. It is applicable only to binary relevance datasets, as recall becomes ambiguous with multiple relevance labels. The formula for Recall@100 is:

$$\text{Recall@100} = \frac{\text{Number of relevant documents in top 100 results}}{\text{Total number of relevant documents}}. \tag{4}$$

Normalized Discounted Cumulative Gain (NDCG) NDCG evaluates the relevance and ranking quality by assigning higher weights to relevant documents appearing earlier in the ranking [**arvelin2002cumulated**]. It is normalized to ensure comparability across queries. NDCG@10 and NDCG@100 indicate the metric calculated for the top 10 and 100 results, respectively. The formula for DCG is:

$$\text{DCG@p} = \sum_{k=1}^{p} \frac{2^{\text{rel}(k)} - 1}{\log_2(k+1)}, \tag{5}$$

where $\text{rel}(k)$ is the relevance score of the document at rank k. NDCG is then calculated as:

$$\text{NDCG@p} = \frac{\text{DCG@p}}{\text{IDCG@p}}, \tag{6}$$

where IDCG@p is the ideal DCG, calculated by sorting the documents in the most relevant order.

3.2 TREC Evaluation Framework

Introduction to TREC Eval TREC Eval is a widely used evaluation framework for computing retrieval metrics such as MAP, MRR, Recall, and NDCG [5]. It processes two main inputs: a *qrel file* (ground truth relevance judgments) and a *run file* (retrieved results from the search system).

PyTREC Eval PyTREC Eval provides a Python interface to TREC Eval, simplifying its integration into search system workflows [6]. By using PyTREC Eval, we can directly calculate the required metrics programmatically, enabling streamlined evaluation and result analysis.

3.3 Dataset Format and Relevance Judgments

Qrel Files Qrel files contain the ground truth relevance judgments for queries. Each qrel file has four columns: 1. **Query ID**: Unique identifier for a query. 2. **Placeholder**: Always `0`, included for format compatibility. 3. **Passage ID**: Unique identifier for a document or passage. 4. **Relevance Label**: Indicates relevance. - *qrels.eval.one.tsv* and *qrels.eval.two.tsv*: Labels range from 0 to 3 (multiple relevance levels). - *qrels.dev.tsv*: Labels are binary (0 = not relevant, 1 = relevant).

Query IDs: `qrels.eval.one.tsv` contains 43 unique query IDs, `qrels.eval.two.tsv` contains 54, and `qrels.dev.tsv` contains 1000. This results in a total of 43+54+1000 = 1097 queries that each system must handle.

Run Files Run files store the retrieval results generated by search systems. Each run file has six columns: 1. **Query ID**: Matches the Query ID in qrel files. 2. **Placeholder**: Typically `Q0`, included for format compatibility. 3. **Passage ID**: ID of a retrieved passage. 4. **Rank**: Position of the passage in the ranked list. 5. **Score**: Score assigned to the passage by the system. 6. **Run Name**: Identifier for the search system producing the results.

3.4 Evaluation Metrics for Specific Datasets

- **qrels.eval.one.tsv and qrels.eval.two.tsv**: Contain multiple relevance labels (0-3). Metrics used for evaluation: - MAP - NDCG@10 - NDCG@100

- **qrels.dev.tsv**: Contains binary relevance labels (0/1). Metrics used for evaluation: - MAP - MRR - Recall@100

3.5 Practical Considerations and Guidelines

- For datasets with multiple relevance labels (`eval.one` and `eval.two`), metrics like NDCG are appropriate due to their ability to handle graded relevance. - For datasets with binary labels (`dev`), metrics like Recall and MRR are emphasized as they effectively measure binary relevance retrieval. - Ensure that the output run files are formatted correctly to enable smooth processing with TREC Eval.

By following these guidelines and employing the appropriate metrics, the evaluation framework ensures accurate assessment of the search systems' performance [7].

4 Results, Performance, and Analysis

4.1 Running Time

The running times for the three systems and the evaluation process are summarized below:

- **System 1:**
  - Query processing: Retrieving the top 500 results for 1097 queries in total from the three `qrel` files takes 6332.62 seconds (˜1 hour 45 minutes).

- **System 2:**
  - Index building: Using `m=6, ef_construct=120`, the process takes 49.48 seconds, and the index file size is 1.60 GB. For `m=10, ef_construct=400`, it takes 193.30 seconds, and the index file size is 1.63 GB.
  - Query processing: Querying, including loading the index into main memory, takes approximately 10 seconds depending on `ef_search` values.
- **System 3:**
  - Query processing: Most of the overhead is due to querying BM25 in System 1. When the `QUERY_BM25` flag is set to `false`, re-ranking from System 1 output files reduces runtime to just a few seconds.
- **TREC_EVAL:**
  - Computing evaluation scores using `trec_eval` takes less than 5 seconds for all systems.

4.2 Overall Systems Comparison

Table 1 presents the evaluation benchmarks for the three systems across the `eval_one`, `eval_two`, and `dev` datasets.

Table 1: Evaluation Benchmarks for the Three Systems. System 1: BM25, System 2: HNSW-based embedding retrieval (m=6, ef_construct=120, ef_search=3000), System 3: BM25 + embedding-based re-ranking (re-ranking top 500 results from System 1 and outputting the top 100 results). MAP: Mean Average Precision, MRR: Mean Reciprocal Rank, Recall@100: Recall for the top 100 results, NDCG@10 / NDCG@100: Normalized Discounted Cumulative Gain for the top 10 and 100 results.

System	Dataset	MAP	MRR	Recall@100	NDCG@10 / NDCG@100
System 1	eval_one	0.3200	-	-	0.4276 / 0.5160
	eval_two	0.2881	-	-	0.4719 / 0.4859
	dev	0.3629	0.3691	0.7765	-
System 2	eval_one	0.4197	-	-	0.6950 / 0.6484
	eval_two	0.4410	-	-	0.6642 / 0.6336
	dev	0.5766	0.5836	0.9174	-
System 3	eval_one	0.4358	-	-	0.7085 / 0.6550
	eval_two	0.4425	-	-	0.6588 / 0.6265
	dev	0.5709	0.5788	0.8583	-

4.2.1 Recall

System 2 achieves the highest recall (˜91.74%), followed by System 3 (˜85.83%), which significantly improves over System 1 (˜77.65%).

Analysis: Embedding-based methods capture semantic relevance, enabling retrieval of documents with similar meaning, even when exact word matches are absent. BM25, a word-by-word matching method, struggles with morphological variations (e.g., *s*, *ing*, *ed*). System 3 improves recall by re-ranking lower-scoring BM25 results based on semantic similarity, surfacing relevant documents initially ranked outside the top 100. System 2's high recall highlights the advantage of embedding-based approaches in capturing semantic nuances.

4.2.2 Mean Reciprocal Rank (MRR)

System 2 and System 3 achieve similar MRR values (˜0.5-0.6), meaning the first relevant result appears in the top two ranks on average. System 1's lower MRR indicates relevant results rank around the third position on average.

Analysis: The improvement in System 3 is due to re-ranking, which pushes semantically relevant documents to the forefront. System 2, leveraging embeddings, inherently captures contextual relevance, yielding competitive MRR values.

4.2.3 Mean Average Precision (MAP)

System 3 marginally outperforms System 2 and significantly improves over System 1. For datasets with multiple relevance labels, semantic embeddings better align with graded relevance levels, as measured by MAP.

Analysis: While System 3's re-ranking leverages embeddings for finer-grained relevance distinctions, System 2 performs comparably due to its robust semantic matching capabilities. System 1 underperforms as it relies solely on word-level matching.

4.2.4 Normalized Discounted Cumulative Gain (NDCG)

System 3 achieves the highest NDCG scores, slightly outperforming System 2. Both systems excel at ranking highly relevant documents earlier, as evidenced by their NDCG@10 values.

Analysis: NDCG emphasizes ranking position, rewarding systems that surface highly relevant documents earlier. System 3 benefits from embedding-based re-ranking, while System 2's embedding index ensures semantic relevance. System 1, constrained by exact word matches, falls behind.

4.3 System Parameter Adjustment and Its Impact

Having compared the performance of the three systems, we now focus on parameter tuning to evaluate how these adjustments affect the performance of Systems 2 and 3. This section provides insights into the trade-offs involved in optimizing the parameters for improved retrieval metrics.

4.3.1 Impact of Parameter Configuration on System 2 Performance

Table 2 summarizes the retrieval metrics for System 2 across different parameter configurations, providing insights into how varying `m`, `ef_construct`, and `ef_search` affects system

performance. Figure 1 visually illustrates these trends, highlighting the relationship between parameter settings and retrieval performance.

Table 2: Evaluation Benchmarks with Different Parameter Configurations for System 2 (HNSW-based embedding retrieval system). ef_c and ef_s are abbreviations of ef_construct and ef_search, respectively.

Dataset	(m, ef_c, ef_s)	MAP	MRR	Recall@100	NDCG@10	NDCG@100
eval_one	(6, 120, 500)	0.4205	-	-	0.6950	0.6494
	(6, 120, 1000)	0.4203	-	-	0.6950	0.6492
	(6, 120, 3000)	0.4197	-	-	0.6950	0.6484
	(10, 400, 3000)	0.4242	-	-	0.6979	0.6525
eval_two	(6, 120, 500)	0.4379	-	-	0.6634	0.6328
	(6, 120, 1000)	0.4372	-	-	0.6634	0.6320
	(6, 120, 3000)	0.4410	-	-	0.6642	0.6336
	(10, 400, 3000)	0.4449	-	-	0.6632	0.6361
dev	(6, 120, 500)	0.5578	0.5652	0.8858	-	-
	(6, 120, 1000)	0.5699	0.5773	0.9048	-	-
	(6, 120, 3000)	0.5766	0.5836	0.9174	-	-
	(10, 400, 3000)	0.5865	0.5936	0.9357	-	-

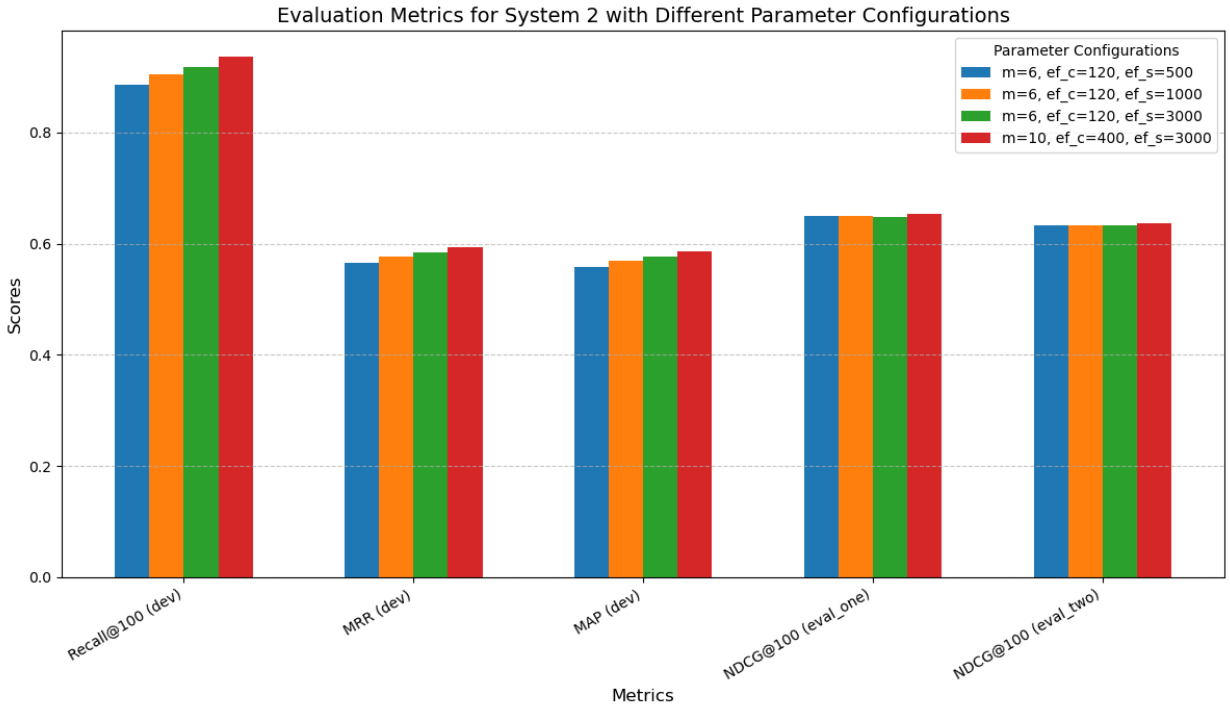

Figure 1: Evaluation Metrics for System 3 with Different Parameter Configurations. Results are visualized based on the data presented in Table 2.

Key Observations:

- **Effect of Increasing `ef_search`:** Increasing `ef_search` from 500 to 3000 consistently improved retrieval performance across all metrics (MAP, MRR, Recall@100, NDCG@10, and NDCG@100). For example, in the `dev` dataset, Recall@100 increased from 0.8858 to 0.9174, and MAP increased from 0.5578 to 0.5766. While this requires slightly higher query processing time, it remains within an acceptable range, with around 10-15 seconds for processing 1097 queries (including index loading time).

- **Effect of Increasing `m` and `ef_construct`:** Increasing `m` from 6 to 10 and `ef_construct` from 120 to 400 during index building slightly improved retrieval metrics. For instance, on the `dev` dataset, Recall@100 increased from 0.9174 to 0.9357, and MAP increased from 0.5766 to 0.5865. However, the index building time also increased significantly, from 49.48 seconds to 193.30 seconds, and the index file size grew marginally from 1.60 GB to 1.63 GB.

- **Trade-offs in Parameter Tuning:** While larger `m` and `ef_construct` values yield higher retrieval accuracy, they increase index building time and memory usage. Similarly, higher `ef_search` improves query accuracy but slightly increases query processing time. These trade-offs should be carefully considered depending on the specific application requirements, such as real-time constraints or storage limitations.

- **Limitations of Embedding-Based Methods:** Although embedding-based methods improve semantic relevance, there may be inconsistencies between embedding similarity and human-labeled relevance scores. This highlights the challenge of aligning embeddings with real-world relevance judgments, which might be influenced by contextual or domain-specific factors.

In conclusion, parameter tuning for System 2 shows clear trends where higher parameter values generally improve retrieval metrics, albeit with diminishing returns and increased resource consumption. The choice of parameters should balance retrieval accuracy and system efficiency.

4.3.2 Effect of Varying Top-K Sizes on System 3 Metrics

Table 3 summarizes the retrieval metrics for System 3 with varying re-ranking configurations, demonstrating how increasing the top-k size influences performance across datasets. Similarly, Figure 2 provides a visual representation of the same data, highlighting the trends and improvements in metrics as the top-k size increases.

Key Observations:

- **Effect of Increasing Re-rank Top Size:** Increasing the number of documents retrieved from System 1 (e.g., from 150 to 500) allows System 3 to better re-rank BM25 results by leveraging semantic embeddings. For instance, on the `eval_one` dataset, MAP increased from 0.4085 to 0.4358, and NDCG@100 increased from 0.6200 to 0.6550. Similarly, on the `dev` dataset, Recall@100 improved from 0.8060 to 0.8583.

- **Time Overhead:** If System 1 results are already available, the additional time required for the re-ranking step is minimal and does not significantly impact processing

Table 3: Evaluation Metrics for System 3 with Different Re-ranking Configurations.

Dataset	Re-rank Top	MAP	MRR	Recall@100	NDCG@10	NDCG@100
eval_one	150	0.4085	-	-	0.6985	0.6200
	300	0.4308	-	-	0.7081	0.6488
	500	0.4358	-	-	0.7085	0.6550
eval_two	150	0.4098	-	-	0.6509	0.5921
	300	0.4313	-	-	0.6591	0.6160
	500	0.4425	-	-	0.6588	0.6265
dev	150	0.5585	0.5669	0.8060	-	-
	300	0.5770	0.5691	0.8450	-	-
	500	0.5709	0.5788	0.8583	-	-

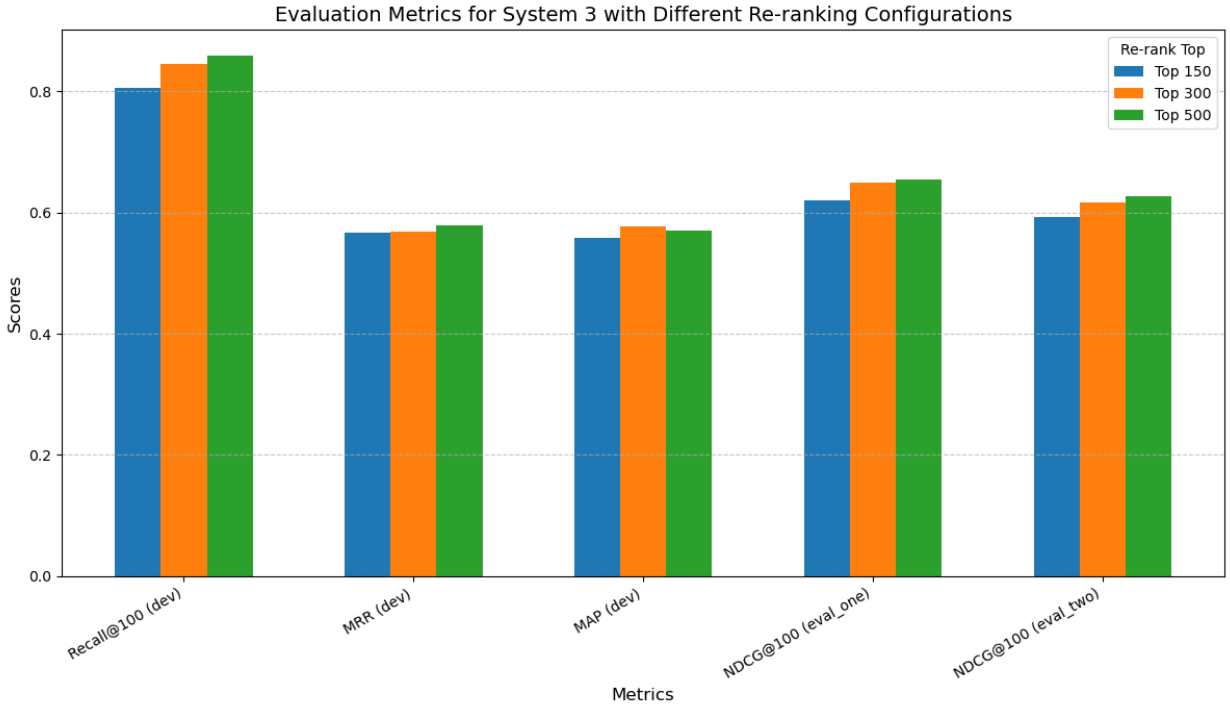

Figure 2: Evaluation Metrics for System 3 with Different Re-ranking Configurations. Results are visualized based on the data presented in Table 3.

efficiency. When retrieving more results from System 1 (e.g., increasing from 150 to 500 top documents), the total query processing time for all 1097 queries increases from 6137.29 seconds to 6332.62 seconds. This corresponds to an average increase per query from 5.59 seconds to 5.77 seconds, which remains within an acceptable range given the potential improvements in retrieval performance.

- **Bottleneck with Larger Top Sizes:** Increasing the number of top documents retrieved beyond 500 leads to diminishing returns and storage inefficiencies. Moreover, such an approach does not effectively address cases where semantic relevance is missed

due to variations in word forms (e.g., differences in *s*, *ing*, or *ed* suffixes). This limitation arises because increasing the size of System 1 results relies on exact matches, which cannot capture semantic variations effectively.

In summary, increasing the number of documents retrieved from System 1 enhances retrieval metrics in System 3 by surfacing semantically relevant documents that might otherwise be overlooked. However, the marginal improvements diminish beyond 500 documents, and further significant gains cannot be achieved solely by increasing the retrieval size.

5 Discussion

The trade-offs among Systems 1, 2, and 3 arise from their differing retrieval mechanisms, reflecting the strengths and limitations of BM25-based and embedding-based approaches. Below is an analysis of their key characteristics and scalability considerations.

5.1 Strengths and Limitations of the Systems

System 1: Efficiency and Flexibility System 1 employs BM25's word-level matching, which provides flexibility for conjunctive and disjunctive queries. This design accommodates variations like suffixes (*s*, *ing*, *ed*), making it suitable for precise keyword searches. Additionally, the final compressed index for 1 million entries is only 124.5 MB, demonstrating its memory efficiency. In theory, optimized implementations of BM25 with inverted indexes can achieve query times as low as 5ms. However, the current implementation of System 1 is not fully optimized and achieves slower query times.

System 2: Semantic Relevance with High Overhead System 2 leverages embedding-based retrieval to encode sentence-level meaning, enabling retrieval of semantically related documents even without term overlap. Its graph-based index requires 1.6 GB for 1 million entries and must be loaded into main memory for queries, leading to significant memory usage. This requirement is due to the inherent design of graph-based indexes, which lack the I/O efficiency of linear structures. For larger datasets, memory limitations could become a bottleneck, and implementing distributed or sharded indexing systems may be challenging.

System 3: Balancing Relevance and Flexibility System 3 integrates BM25-based initial ranking with embedding-based re-ranking, combining speed and semantic accuracy. This hybrid approach improves the retrieval of semantically related documents compared to System 1. However, its reliance on the top-k results from System 1 limits its ability to retrieve documents excluded from the initial retrieval due to term mismatches.

5.2 Embedding vs. BM25: A Comparative Analysis

Accuracy vs. Flexibility: Embedding-based methods excel in capturing semantic relationships, particularly in queries with no exact term overlap. BM25, with its word-level matching, is more flexible for handling variations like plural forms and suffixes, enabling precise keyword searches. However, BM25 does not capture the broader semantic context of queries.

Memory Efficiency: BM25's inverted index is efficient, using a linear structure that allows relative positions to be computed directly without loading the entire index into memory. For example, System 1's compressed index for 1 million entries is only 124.5 MB. In contrast, embedding-based systems like System 2 require dense vector representations and graph structures, resulting in significantly higher memory usage (e.g., 1.6 GB for the same dataset size).

Precision Queries: BM25 is particularly effective in contexts requiring precise or conjunctive queries, such as legal or medical searches. Embedding-based methods, while better suited for general semantic retrieval, are less efficient for tasks demanding exact matches.

5.3 Indexing and Scalability

BM25 and Inverted Index: System 1's inverted index is scalable and I/O-efficient, enabling sub-millisecond query times with proper optimizations. Its linear design makes it well-suited for distributed or sharded implementations, enhancing scalability for large datasets.

Embedding-Based Graph Index: System 2's graph-based index requires the entire structure to be loaded into memory, limiting scalability for large-scale retrieval tasks. The design of graph-based systems poses challenges in achieving the same level of I/O efficiency as linear indexes, which hinders their application in memory-constrained environments.

5.4 Concluding Remarks

System 1 is the most efficient in terms of memory and query speed, making it ideal for keyword-driven applications. However, its ability to capture semantic relevance is limited. System 2 achieves higher semantic accuracy through embedding-based retrieval but requires substantial memory resources and is less scalable for larger datasets. System 3 balances the trade-offs by combining the strengths of both approaches, improving semantic relevance while maintaining acceptable efficiency. Despite this, its reliance on System 1's initial retrieval set constrains its ability to fully address semantic mismatches in the ranking phase.

6 Conclusion

The comparative analysis of three retrieval systems highlights the trade-offs between retrieval accuracy and system efficiency. System 1, based on BM25, excels in computational efficiency and memory usage but struggles with semantic relevance. System 2, leveraging dense embeddings and the HNSW algorithm, achieves superior semantic retrieval but at the cost of increased memory and computational overhead. System 3 effectively balances these trade-offs by integrating BM25 for initial retrieval and embedding-based re-ranking for enhanced semantic accuracy.

While System 3 provides the most balanced performance, its reliance on BM25 for initial ranking imposes limitations in cases where term mismatches exclude relevant documents from the candidate set. Overall, this study underscores the importance of selecting retrieval methods aligned with specific application requirements and resource constraints.

References

[1] Yu A. Malkov and D. A. Yashunin. "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 42.4 (Apr. 2020), pp. 824–836. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2018.2889473. URL: https://doi.org/10.1109/TPAMI.2018.2889473.

[2] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[3] Ellen M. Voorhees and Dawn M. Tice. "Building a question answering test collection". In: *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '00. Athens, Greece: Association for Computing Machinery, 2000, pp. 200–207. ISBN: 1581132263. DOI: 10.1145/345508.345577. URL: https://doi.org/10.1145/345508.345577.

[4] Stephen E Robertson. "The probability ranking principle in IR". In: *Journal of documentation* 33.4 (1977), pp. 294–304.

[5] EM Voorhees. *TREC: Experiment and evaluation in information retrieval*. 2005.

[6] Christophe Van Gysel and Maarten de Rijke. "Pytrec_eval: An Extremely Fast Python Interface to trec_eval". In: *SIGIR*. ACM, 2018.

[7] Chris Buckley and Ellen M. Voorhees. "Evaluating evaluation measure stability". In: *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '00. Athens, Greece: Association for Computing Machinery, 2000, pp. 33–40. ISBN: 1581132263. DOI: 10.1145/345508.345543. URL: https://doi.org/10.1145/345508.345543.