Web Search Engines Final Project: Evaluation of Search Ranking Methods

Henry Kam, Jeffery Zeng

December 25, 2024

1 Instructions

General Information

The code for this report is available at (https://github.com/gulpinhenry/search-ranking-eval).

1.0.1 To run the code

This was tested on torch=2.4.1 and cuda=12.1. Please run 'pip install -r requirements.txt'. To generate embeddings, please first encode the passages (https://github.com/gulpinhenry/search-ranking-eval/blob/main/embedding/encode_passages.py. Then you can test different pipelines in the pipelines directory. For more details on code structure, please refer to section 6.

2 Introduction

Evaluating the efficiency and effectiveness of different search ranking architectures have been a vital part in determining the success of various modern-day applications. From the specific word boolean retrieval systems to the probablistic ranking function methods, cross-encoders and bi-encoders have become core approaches for encoding and retrieving information. Cross-encoders jointly encode query and document pairs, allowing for intricate interactions that can enhance retrieval accuracy but also increase computational complexity. In contrast, bi-encoders independently encode queries and documents, offering advantages in scalability and speed especially in large-scale retrieval scenarios.
The MS MARCO (Microsoft MAchine Reading COmprehension) (Bajaj et al., 2018) dataset has become a classical benchmark in evaluating information retrieval models, as it contains a diverse and extensive collection of real-world queries and passages. Its' vast nature makes it an ideal platform for assessing the performance of various encoding strategies. Using this dataset can allows researchers to benchmark models against a standardized set of tasks, which is particularly useful in our case.
To further enhance retrieval performance, efficient indexing and search algorithms are indispensable. Hierarchical Navigable Small World (HNSW)(Malkov & Yashunin, 2018) graphs have gained traction as a state-of-the-art method for approximate nearest neighbor (ANN) search, offering a balance between speed and accuracy. HNSW structures, which are inspired by skip lists, facilitate faster traversal and retrieval in high-dimensional spaces, making them particularly suitable for handling the dense embeddings produced by both cross-encoders and bi-encoders. By integrating HNSW graphs with these encoding models, it can be possible to achieve swift and scalable retrieval without significant compromises on effectiveness.
This study aims to comprehensively evaluate the performance of various cross-encoder and bi-encoder models within the context of the MS MARCO dataset, also attempting to utilize HNSW graphs for efficient indexing and retrieval. The primary objective is to assess how these models perform across several critical metrics: processing time, embedding size, and retrieval rankings. Processing time is a crucial factor, especially for online scenarios. Embedding size is a necessary metric to evaluate in terms of scalability and processing

speed. Retrieval effectiveness, often measured through metrics such as precision, recall, and mean reciprocal rank, remains the cornerstone of evaluating the practical utility of any information retrieval system.
By systematically measuring these metrics, this report seeks to identify different strengths and limitations of cross-encoder and bi-encoder models when paired with HNSW graphs. By understanding these different metrics we are able to evaluate which scenarios are more fitting for each model.

3 Related Works

Information retrieval (IR) has undergone significant transformations from applying advanced machine learning models and using efficient indexing techniques. This section reviews the foundational concepts in IR, the evolution of retrival models, the integration of neural architectures, and the utilization of advanced indexing structures such as Hierarchical Navigable Small World (HNSW) graphs. Additionally, explains previosly existing evaluations using the MS MARCO dataset.
Information retrievals provide a mathematical basis for different search processes. Early on, Boolean retrieval models and the vector space models (VSM) laid the groundwork to create basic ideas such as term matching and high dimensional vector space document representation (Manning et al., 2008). Boolean retrieval was used a while ago, as it was a simple yet efficient approach using logical boolean operators to retrieve documents. However, its' performance ultimately relied on the user's proficiency to search the right keywords, as it required direct term matching and struggled to handle complex queries. However, the VSM allowed documents to be represented semantically in high dimensional vector spaces, allowing ranking to be retrieved via similarity measures like cosine similarity (Manning et al., 2008), setting the basis for future models.
Probabilistic models, such as BM25 and language models, further created more insights to estimiate relevance of documents using bayesian probability (He, 2009). BM25, a widely adopted ranking function, uses parameters including term frequency and inverse document frequency that can be precomputed to retrieve relevant results via impact score, allowing it to be relatively performant while being compute-friendly. Other language models including the query likelihood or bag of words n-gram model also offer a probabilistic interpretation of queries, where they calculate the probability of generating queries using the document's contextual language model (Jurafsky & Martin, 2009).
By integration these models together into IR has created more opportunities to increase retrieval effectiveness. This includes algorithms such as the Rocchio algorithm, utilizing relevance feedback to tune query vectors to maximize the opportunity of obtaining relevant documents while minimizing the likelihood of retrieving non-relevant queries (Manning et al., 2008). This continued after with Learning to Rank (LTR) methods, where methods use reinforcement and supervised learning techniques to optimize these ranking functions (Jurafsky & Martin, 2009). Ranking Support Vector Machines are classification models that rank documents by creating hyperplanes between relevant and irrelevant queries, then maximizing the margins between the two (Do et al., 2017). Furthermore, models can also incorporate multimodal data, from images, javascript, and other features to create better relevance feedback and rank scoring.
Other advancements in neural network backbones, particularly transformers have also

created new opportunities to revolutionize IR. Cross-encoders and bi-encoders have utilized this to create more nuanced understanding of documents. Cross-encoders jointly encode queries and documents, allowing for more richer interactions and higher retrieval accuracy. However, this comes with the need for running these cross-encoders with each document for evaluation, causing large computational overhead (Nogueira & Cho, 2020). Bi-encoders on the other hand, independently encode queries and documents into their own latent space, allowing for a more scalable and efficient retrieval approach as the embeddings can be precomputed (Karpukhin et al., 2020). However, as a result the resulting ranking performance may not be as good as cross-encoders (Karpukhin et al., 2020).
Searching for closely related queries within a vector space is also a very deeply research area. This problem is coined as a nearest neighbors problem, where for large scale IR systems, this is really important. Hierarchical Navigable Small World (HNSW) graphs have emerged as a state-of-the-art method for approximate nearest neighbor search, balancing speed and accuracy (Malkov & Yashunin, 2018). HNSW structures motivated by skip lists facilitate rapid traversal in high-dimensional embedding spaces, making them particularly suitable for handling the dense representations produced by both cross-encoders and bi-encoders. Their hierarchical nature allows for efficient navigation and scalability, essential for real-time applications and large datasets. Integrating HNSW with neural embedding models has shown promising results in enhancing retrieval effectiveness without compromising on processing time, thereby supporting scalable and efficient IR systems (Formal et al., 2021).
The MS MARCO dataset has become a benchmark for evaluating these neural retrieval models due to its extensive and diverse collection of real-world queries and passages. Studies utilizing this dataset have demonstrated the effectiveness of transformer-based models in capturing semantic relevance and improving retrieval metrics (Craswell et al., 2020).
Evaluating IR systems involves assessing multiple dimensions, including processing time, embedding size, and retrieval effectiveness. Processing time is crucial for ensuring low latency in real-time applications, while embedding size impacts storage requirements and computational resources. Retrieval effectiveness, typically measured through metrics like precision, recall, and mean reciprocal rank (MRR), remains arguably most important of evaluating the practical utility of IR systems (Manning et al., 2008).
Studies have consistently shown that advancements in retrieval models and indexing techniques directly correlate with improvements in these evaluation metrics. For instance, the transition from traditional probabilistic models to transformer-based neural models has significantly enhanced retrieval effectiveness on benchmark datasets like MS MARCO, albeit with increased computational demands (Karpukhin et al., 2020).
Our study builds upon these foundations by evaluating the performance of various cross-encoder and bi-encoder models within the MS MARCO dataset, utilizing HNSW graphs for indexing. We measure different metrics across speed, size, and ranking performance to evaluate which types of models are suited for various scenarios.

4 Dataset

We used the MSMARCO dataset, as it contains a variety of passages and is widely accepted as the baseline for different evaluation experiments. We used a subset of MS-

MARCO to reduce training and evaluation time, as we utilized HW3's MSMARCO subsets.

5 Our Contribution

1. We systematically assess various sentence-transformer models (Reimers & Gurevych, 2019), and test it on the MSMARCO datset. We explore how these different models perform on the MSMARCO datasets, all while also integrating certain embeddings with HNSW for ANN retrieval.

2. In addition to neural models we also evaluate the BM25 model enhanced with different cascaded reranking strategies. By implementing different types of cascading, we are able to see how we can balance the trade offs of the speed of the bi-encoder and the performance of the cross-encoders.

3. We conduct an in-depth analysis of several critical metrics, including processing time, embedding size, and retrieval effectiveness. Through this we can infer the scalability and performance of these models in different application settings.

4. By integrating both sentence-transformer models and BM25-based methods with HNSW graphs, we demonstrate the feasibility of achieving rapid and scalable approximate nearest neighbor (ANN) search in high-dimensional embedding spaces. Through this, we are able to gain a better understanding of how HNSW and other fast ANN algorithms can impact other results

In summary, our contributions advance the understanding of how modern neural retrieval models and traditional probabilistic methods perform in conjunction with efficient indexing techniques like HNSW graphs. By evaluating these models across multiple dimensions, we offer valuable guidance for developing scalable and effective information retrieval systems tailored to diverse application requirements.

6 Code Structure

The implementation of our information retrieval system is meticulously organized into modular components, each encapsulating distinct functionalities essential for efficient and effective retrieval operations. We have examined each part of the ranking pipeline, from creating embeddings to creating different indexes and running the query processing. We have implemented this in python and created our own automated pipeline where we can swap out different modules and models for different ones quickly and efficiently.

6.1 Overview of Modules

Our codebase is partitioned into five primary modules, each responsible for specific aspects of the retrieval system:

1. **Utilities Module (`utils`)**

2. **Indexers Module** (`indexers`)
3. **Retrieval Pipeline Module** (`retrieval_pipeline`)
4. **Encoding Script** (`encode_passages`)
5. **Main Execution Script** (`main`)

6.2 Utilities Module (`utils`)

The `utils` module serves as the foundational layer of the system, providing essential classes and functions that facilitate data management and inter-component communication.

6.2.1 PassageManager

Purpose: Manages the loading, storage, and retrieval of passage data and their corresponding embeddings.
Responsibilities:

- Load passage texts and embeddings from specified files.
- Provide methods to retrieve passage texts and embeddings based on document IDs.
- Maps document Ids to their embeddings and passages

6.2.2 QueryManager

Purpose: Handles the management of queries, including their retrieval and embedding.
Responsibilities:

- Load query texts and, if applicable, their embeddings from designated files.
- Provide methods to retrieve query texts and embeddings based on query IDs.

6.2.3 BaseIndexer

Purpose: Serves as an abstract base class for all indexing strategies.
Responsibilities:

- Define a standard interface (`search` method) that all indexers must implement.
- Facilitate uniform interaction with different indexing mechanisms.

6.3 Indexers Module (`indexers`)

The `indexers` module encapsulates various indexing and retrieval strategies through specialized classes, each implementing a unique approach to document ranking and retrieval.

6.3.1 CrossEncoderIndexer

Purpose: Implements a cross-encoder-based retrieval mechanism.
Responsibilities:

- Initialize with a pre-trained cross-encoder model.
- Perform joint encoding of queries and documents to assess relevance.
- Rank documents based on the cross-encoder's scoring.

6.3.2 CosineSimilarityIndexer

Purpose: Implements a bi-encoder-based retrieval mechanism using cosine similarity.
Responsibilities:

- Retrieve query and document embeddings independently.
- Compute cosine similarity scores between query and document embeddings.
- Rank documents based on similarity scores.

6.3.3 BM25Indexer

Purpose: Implements the BM25 probabilistic retrieval model.
Responsibilities:

- Calculate BM25 scores for documents based on term frequency and inverse document frequency.
- Rank documents according to their BM25 scores.

6.3.4 HNSWIndexer

Purpose: Implements efficient approximate nearest neighbor (ANN) search using Hierarchical Navigable Small World (HNSW) graphs.
Responsibilities:

- Build and maintain an HNSW index using precomputed document embeddings.
- Perform rapid similarity searches within the HNSW graph.
- Retrieve top-k similar documents based on ANN search results.

6.4 Retrieval Pipeline Module (`retrieval_pipeline`)

The `retrieval_pipeline` module orchestrates the sequential application of multiple indexers to refine and enhance retrieval results through cascaded reranking.

6.4.1 RetrievalPipeline

Purpose: Manages the integration of various indexers into a cohesive retrieval workflow.
Responsibilities:

- Initialize with a list of indexers representing different retrieval strategies.
- Sequentially apply each indexer to narrow down and rerank the pool of candidate documents.
- Aggregate and return the final ranked list of documents based on cumulative scores from all indexers.

6.5 Encoding Script (`encode_passages`)

The `encode_passages` script is dedicated to preprocessing and encoding passage data into latent space using SentenceTransformers models, used for efficient similarity-based retrieval.

6.5.1 Passage Encoding Process

Purpose: Encode textual passages into dense vector representations suitable for similarity computations.
Responsibilities:

- Load passage texts from collection and subset files.
- Utilize a specified SentenceTransformers model to generate embeddings for passages in batches.
- Save the encoded embeddings to a pickle file for subsequent retrieval operations.
- Ensure efficient utilization of computational resources by leveraging GPU acceleration if available.

6.6 Example Execution Script (`example`)

The `example` script serves as the entry point for executing retrieval tasks, integrating all components to perform searches and display results. This is an example of how to use our pipeline to insert different models in different orders for a cascading mechanism

6.6.1 Main Execution Flow

Purpose: Initialize managers and indexers, perform retrieval operations, and facilitate result exploration.
Responsibilities:

- Initialize instances of `PassageManager` and `QueryManager` with appropriate data files.

- Instantiate indexers, such as `CosineSimilarityIndexer` and `CrossEncoderIndexer`, based on the retrieval strategy.
- Execute search operations for specified queries, retrieving and displaying top-ranked documents.
- Optionally, employ interactive exploration tools to analyze and debug retrieval results.

6.7 Interaction Between Components

The seamless interaction between modules ensures a robust and efficient retrieval process:

1. **Data Management:**
   - The `PassageManager` and `QueryManager` load and manage the passage and query data, respectively, providing necessary information to the indexers during retrieval.
2. **Indexing and Retrieval:**
   - Indexers access passage and query embeddings through the managers to perform their specific retrieval computations.
   - Each indexer processes queries and documents according to its unique strategy, outputting ranked lists of relevant documents.
3. **Retrieval Pipeline:**
   - The `RetrievalPipeline` integrates multiple indexers, allowing for initial retrieval using a fast method (e.g., cosine similarity) followed by more precise reranking (e.g., cross-encoder), thereby enhancing overall retrieval effectiveness.
4. **Encoding Passages:**
   - The `encode_passages` script is executed prior to retrieval to ensure that all passages are encoded and ready for similarity computations, thereby enabling efficient retrieval operations.
5. **Example Execution:**
   - The `example` script ties together data management, indexing, and retrieval pipeline components, facilitating the execution of end-to-end retrieval tasks and the display of results.

6.8 Integration with HNSW Graphs

The incorporation of Hierarchical Navigable Small World (HNSW) graphs is managed through the `HNSWIndexer` class, which leverages FAISS (**F**acebook **AI Similarity Search**) to build and query the HNSW index. This integration enables efficient approximate nearest neighbor searches within high-dimensional embedding spaces, significantly reducing retrieval latency while maintaining high accuracy.

6.8.1 Building the HNSW Index

- The `HNSWIndexer` retrieves all passage embeddings from the `PassageManager`.
- Embeddings are normalized to facilitate inner product similarity measures.
- An HNSW index is constructed using FAISS, with parameters such as `m`, `ef_construction`, and `ef_search` tuned for optimal performance.
- The index is populated with the normalized embeddings, enabling rapid similarity searches.

6.8.2 Performing Searches with HNSW

- For a given query, the `HNSWIndexer` retrieves the corresponding query embedding from the `QueryManager`.
- The query embedding is normalized and used to perform a similarity search within the HNSW index.
- The top-k similar documents are retrieved based on the search results, facilitating efficient and scalable retrieval operations.

7 Results

The result of our system is shown in Table 1:

Table 1: Evaluation Metrics for Query Relevance (result*100)

Metric	**qrel.eval.one.tsv**	**qrel.eval.two.tsv**	**qrels.dev.tsv**
BM25 + Bi-encoder (MiniLM)			
MRR@10	77.71	83.33	33.54
Recall@100	33.98	33.50	50.17
NDCG@10	19.58	27.67	-
NDCG@100	39.36	41.26	-
MAP@10	-	-	32.73
MAP@100	-	-	33.03
BM25 + CrossEncoder (MiniLM)			
MRR@10	82.25	83.64	37.13
Recall@100	36.08	35.96	50.40
NDCG@10	21.19	29.70	-
NDCG@100	42.28	44.50	-
MAP@10	-	-	36.24
MAP@100	-	-	36.50

Continued on next page

Metric	**qrel.eval.one.tsv**	**qrel.eval.two.tsv**	**qrels.dev.tsv**
BM25 + CrossEncoder (TinyBERT)			
MRR@10	80.62	81.94	34.40
Recall@100	35.03	34.71	49.92
NDCG@10	20.42	27.71	-
NDCG@100	40.84	42.02	-
MAP@10	-	-	33.72
MAP@100	-	-	33.99
HNSW			
MRR@10	91.57	93.67	55.14
Recall@100	60.05	64.48	87.35
NDCG@10	29.52	36.55	-
NDCG@100	61.54	64.61	-
MAP@10	-	-	54.48
MAP@100	-	-	55.16
HNSW + CrossEncoder (MiniLM)			
MRR@10	98.84	94.34	63.32
Recall@100	63.65	69.04	89.05
NDCG@10	33.86	40.62	-
NDCG@100	68.12	70.09	-
MAP@10	-	-	62.59
MAP@100	-	-	63.00
HNSW + CrossEncoder (TinyBERT)			
MRR@10	95.35	90.06	57.61
Recall@100	61.70	66.14	88.52
NDCG@10	31.22	36.04	-
NDCG@100	64.65	64.99	-
MAP@10	-	-	56.96
MAP@100	-	-	57.55
HNSW + Bi-encoder (MiniLM)			
MRR@10	91.57	93.67	55.71
Recall@100	58.66	64.52	89.05
NDCG@10	28.78	36.55	-
NDCG@100	60.72	64.58	-
MAP@10	-	-	54.99
MAP@100	-	-	55.68

8 Conclusion

We can draw the following conclusions based on the results:

- HNSW with CrossEncoder (MiniLM) demonstrates the highest performance across most metrics, particularly in Mean Reciprocal Rank (MRR@10), achieving a score of 98.84 for `qrel.eval.one.tsv`. This indicates excellent retrieval capability for the top 10 results.
- Compared to HNSW, BM25 tends to miss some relevant results, especially those that are semantically different.
- Cross-Encoders generally outperform Bi-Encoders due to their more sophisticated comparison methods, which take additional contextual factors into account.
- HNSW performs comparably to cosine similarity methods, indicating its effectiveness in retrieving nearest neighbors with minimal information loss.
- The performance of different Cross-Encoders varies according to the specific model used, highlighting the importance of model selection in retrieval effectiveness.

9 Future Work

Building on the insights gained from this study, future research could explore the following directions:

1. **Model Optimization:** look into other techniques to further optimize different neural models to enhance processing times without needing to compromise retrieval effectiveness. We can also look at different hyperparameter tunings to extract the most out of the models
2. **Scalability:** exploring more advanced indexing structures and parallel processing methods like mapreduce and elasticsearch to eventually perform this on the entire MSMarco dataset
3. **User Feedback:** implement user behavior and feedback into evaluation metrics to better align retrieval ranking with user needs and preferences

References

Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., & Wang, T. (2018). Ms marco: A human generated machine reading comprehension dataset. https://arxiv.org/abs/1611.09268

Craswell, N., Mitra, B., Yilmaz, E., Campos, D., & Voorhees, E. M. (2020). Overview of the trec 2019 deep learning track. https://arxiv.org/abs/2003.07820

Do, P.-K., Nguyen, H.-T., Tran, C.-X., Nguyen, M.-T., & Nguyen, M.-L. (2017). Legal question answering using ranking svm and deep convolutional neural network. https://arxiv.org/abs/1703.05320

Formal, T., Lassance, C., Piwowarski, B., & Clinchant, S. (2021). Splade v2: Sparse lexical and expansion model for information retrieval. https://arxiv.org/abs/2109.10086

He, B. (2009). Probability ranking principle. In L. LIU & M. T. ÖZSU (Eds.), *Encyclopedia of database systems* (pp. 2168–2169). Springer US. https://doi.org/10.1007/978-0-387-39940-9_930

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing : An introduction to natural language processing, computational linguistics, and speech recognition.* Pearson Prentice Hall. http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. https://arxiv.org/abs/2004.04906

Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. https://arxiv.org/abs/1603.09320

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval.* Cambridge University Press. http://nlp.stanford.edu/IR-book/information-retrieval-book.html

Nogueira, R., & Cho, K. (2020). Passage re-ranking with bert. https://arxiv.org/abs/1901.04085

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing.* https://arxiv.org/abs/1908.10084