Dynamic Tiered Indexing for Scalable Sparse Neural Retrieval Systems

Abhinav Gupta
New York University

Roman Vakhrushev
New York University

***Abstract*—Sparse Neural Retriever models like DeepImpact have shown great promise in improving search result accuracy, but their large index sizes as well as slow retrieval speed can make them impractical for many applications. In this paper, we explore the use of pruning techniques to reduce the size of the DeepImpact index while maintaining its effectiveness. We propose and evaluate several scoring strategies, including document counts, inverse logarithm score, and inverse bin score, and demonstrate that our approach can achieve significant reductions in index size and speed while preserving competitive retrieval performance. Our results demonstrate that pruning can be an effective solution for mitigating the efficiency limitations of DeepImpact, making it a more viable option for large-scale search applications.**

I. Introduction

The rapid growth of digital data has led to an increasing demand for efficient and effective information retrieval systems. These systems aim to retrieve relevant information from large collections of data, and their performance has a significant impact on various applications, including search engines, recommender systems, and question answering systems. Traditional information retrieval models, such as BM25, have been widely used due to their simplicity and efficiency. However, they often struggle to capture the complex relationships between query terms and document content, leading to suboptimal retrieval performance.

Recently, sparse neural retrievers, such as DeepImpact [Mallia et al., 2021], have emerged as a promising approach to improve the accuracy of information retrieval systems. These models leverage deep learning architectures to capture the semantic relationships between queries and documents, leading to more accurate and relevant search results. However, sparse neural retrievers often come with a significant computational cost, which can make them impractical for large-scale search applications. In particular, the large index sizes and slow retrieval speeds of these models can limit their adoption in real-world scenarios.

To address these challenges, we focus on developing efficient pruning techniques for sparse neural retrievers, with the goal of reducing their computational cost while maintaining their effectiveness. By pruning the index of a sparse neural retriever, we aim to reduce its memory footprint and computational requirements, making it possible to handle larger datasets and more complex queries. In this paper, we explore the application of pruning techniques to DeepImpact, a state-of-the-art sparse neural retriever, and investigate the effectiveness of various scoring strategies in reducing the index size while preserving competitive retrieval performance.

A. DeepImpact

DeepImpact is a novel neural ranking model designed to improve the quality of retrieval in search systems [Mallia et al., 2021]. Unlike traditional methods such as BM25, which rely on term-frequency inverse document frequency (TF-IDF) and other heuristics, DeepImpact employs a deep learning architecture to capture the complex relationships between query terms and document content. This approach enables DeepImpact to better understand the context and intent behind a query, leading to more accurate and relevant search results, and resolving the vocabulary mismatch problem.

The DeepImpact model works by utilizing the query expansion procedure followed by a neural network. First, the model expands documents by injecting new and rewriting the frequency of old terms. Then, both the injected and original terms are fed into contextual LM encoder. The first occurrence of each unique term is then provided as input to the impact score encoder, which is a two-layer multi-layer perceptron (MLP) with ReLU activations. As a result, a single-value score for each unique term in the document is produced, which represents its impact. Given a particular query, the model computes a score for each document as simply the sum of impacts for the intersection of terms in the document and the query.

Even though the DeepImpact architecture demonstrates very good results and solves the vocabulary mismatch problem, it often runs significantly slower for search compared to standard BM25-based index. In particular, Mean Response Time as reported in [Mallia et al., 2021] can be up to 6 times larger compared that of BM25 index. So, one possible idea, which we tried to implement, is to prune the DeepImpact index to improve its running time.

B. Pruning

Information retrieval systems are designed to efficiently retrieve relevant information from large collections of data. One of the key challenges in building effective information retrieval systems is managing the complexity and scale of the data, while also providing fast and accurate query processing. To address this challenge, various indexing and pruning techniques have been developed to reduce the computational cost and improve the efficiency of retrieval systems. Among

these techniques, static pruning has emerged as a promising approach to improve the performance of information retrieval systems. Static pruning involves pre-computing and storing a subset of the index that contains the most relevant and frequently accessed information, while discarding the less relevant parts of the index. By doing so, static pruning reduces the memory footprint and computational requirements of the retrieval system, making it possible to handle larger datasets and more complex queries.

There are many different methods for static pruning, the methods usually work by removal of postings, documents, or terms from the index. In particular, in paper [Anagnostopoulos et al., 2015] the authors implement several greedy algorithms to do document and posting-based pruning, the paper [Rodriguez and Suel, 2018] extends these methods and uses these features as well as some other features to learn optimal pruning schemes for BM25-based index via machine learning. These and some other methods can also be applied to sparse neural retrievers, the paper [Lassance et al., 2023] demonstrates how this could be done and the results this approach could achieve.

II. Methodology

Our methodology for this study involves several key steps: index building, pruning procedures, scoring strategies, and evaluation. We aim to compare the performance of BM25, DeepImpact, and pruned DeepImpact indexes in terms of query relevance and search efficiency.

A. Index Building and Pruning Procedure

The first step before the pruning can be applied is building the indices. We start by creating the indices using the datasets described in the corresponding sections. Both the index and the original collection it was built on should be kept for later pruning. We then have to search the index using the train dataset to obtain the top-1000 results for each query. Based on the results we obtain we have to select the best documents using the appropriate scoring schemes. For each document, we computed scores in three different ways (see the sections below). The scores are then sorted in descending order. Given a percentage of documents we want to keep in the index p (i.e. $p = 25\%$) and the number of documents in the collection d, we selected top $p \times d$ documents using the scores we computed. The documents that were selected by the pruning scheme have to be fetched from the original collection and combined in a single (or multiple) file that is then used as a new collection. The index is then rebuilt using this collection. In the following sections we discuss different ways to compute the scores for documents.

B. Scoring Strategies

1) Document Counts: One of the simpler methods to compute the scores for documents is by document counts (doc counts). For each document, we simply count the number of times this documents appears in the top-1000 results generated from training set. Documents with larger doc count score are considered to be more important and are kept in the pruning process.

The motivation for this method is the following: assuming the training query distribution will be the same as the testing query distribution, and the original index find optimal or near-optimal set of documents for each query, we want to select documents that appear in as many query results as possible. That way, given the query to the original index and pruned index, we can expect the output from the pruned index to have as many documents from the original index as possible. The downside of this approach is the fact that we do not account for ranks of the returned results. Two documents with the same doc count may, in fact, have very different level of importance: one document may always appear in top-10 results, the other one may never get to top-10. To address this issue, we also considered two other scoring strategies.

2) Inverse Logarithm Score: Another way to compute the scores is by inverse logarithm score (log score). In this scenario, we also consider ranks of the documents (1-1000 for top-1000) we obtain after evaluating the original index using the train query set (computing top 1000 documents for each query). Then, given rank r of a document for some particular query, its score per query is computed as follows: $\frac{1}{\log_2 r+1}$. Then for a given document d_i, all the scores are added up to compute the aggregate score $S(d_i) = \sum_q \frac{1}{\log_2 r_q+1}$, where q is a query from a training set and r_q is the rank of a document in the top-1000 results for query q (if a document did not appear in the result, its score is not updated). The method is inspired by NDCG metric and it weighs the different ranks differently. In fact, the score per query ranges from 1 (for top-1 result) to ≈ 0.1(for top-1000 result). Even though the scores are smoothed by the logarithm function to allow for closer scores for top-1 and top-1000 results, the penalty for being a top-1000 result may still be a little too harsh depending on the application: a document that only appeared in one search result with rank 1 is weighted higher than a document that appeared in 9 search results with rank-1000. So, we also considered another variation to this scoring strategy as discussed in the next section.

3) Inverse Bin Score: The inverse bin score (bin score) is based on the paper [Rodriguez and Suel, 2018]. Instead of directly computing the scores based on ranks, we compute them based on bins as shown in table I. We then use the same scoring formula as in inverse logarithm score, so that for document d_i, the aggregated score is $S(d_i) = \sum_q \frac{1}{\log_2 b_q+1}$, where q is a query from a training set and b_q is the bin number computed based on ranking using table I. Unlike the inverse logarithm score, this scoring strategy results in higher scores for all documents with rank larger than 11. Also, it differentiates less between documents with bad ranks (i.e. ranks 641 and 1000 would get the same score of $\frac{1}{\log_2(17+1)} \approx 0.24$). That way, the smallest score a document can get per query is ≈ 0.24, which is almost 25% of the score for top-1 document.

Although this idea can be applicable in some settings, it demonstrated very similar results to log score in our evaluation, so we decided not to include the results for it in the final

TABLE I: Rank-Bin Partitioning

Rank	Bin
1–10	1–10
11–20	11
21–40	12
41–80	13
81–160	14
161–320	15
321–640	16
641–1000	17

report.

III. Datasets

We used several different datasets for different parts of the project. In this section, we discuss what these datasets are and how we used them to do the pruning.

A. Index Building

We used the augmented MS Marco passage ranking dataset to build the original DeepImpact index to be pruned. This dataset is the version of MS Marco passage dataset processed by DeepImpact: in particular, it has already gone through document expansion and term reweighting. We also used the standard MS Marco passage dataset to build the BM25 index (for evaluation and comparison). Both versions of the dataset contain about 8.8 million documents of various size and contents (the documents and their IDs are the same in both datasets). In addition to documents, the DeepImpact version for each document also contains a vector of impact scores for different terms, which is used in the retrieval.

B. Training

We also used a query training dataset to learn the importance of different documents in the index. For this purpose, we used a TREC2020 version of the training query dataset [Craswell et al., 2020], which contains about 800,000 queries. Instead of using the whole set of 800,000 queries, which was taking too much time and resources in our system, we randomly sampled 200,000 queries for this purpose. The 200,000 query subset was used to compute scores for the documents with all the scoring strategies discussed in the corresponding sections.

C. Evaluation

For the purposes of evaluation, we used two different datasets. The first dataset is a test dataset from TREC2020 [Craswell et al., 2020], which only has 200 queries. The second dataset is the dev small dataset, having 6980 queries. Both datasets had appropriate qrel documents, which we used in the evaluation process.

IV. Experiments

To test our proposed method, we used Pyserini [Lin et al., 2021]. Pyserini is an open-source, Python-based toolkit for efficient and reproducible search evaluation. It provides a flexible and modular framework for indexing, searching, and evaluating large-scale information retrieval systems. Pyserini is designed to support a wide range of search tasks, including interactive retrieval, query ranking, and passage retrieval, and is compatible with various indexing and retrieval libraries, such as Anserini and Lucene. We utilized the index building, the search, and the evaluation functions and scripts provided by Pyserini for our experiments.

The index building step for large indices (based on 8.8 million passage datasets) took about 1.5 hours. The index building for smaller (pruned) datasets took time proportional to the size of pruned datasets. Running evaluations on the training dataset (200,000 queries) using DeepImpact index took 4 hours.

We also demonstrate the evaluation on different pruned indices we built and organized. All experiments were conducted on a laptop equipped with an Intel Core i5-8300H CPU @ 2.30GHz and 16 GB of RAM, running Microsoft Windows 11 Home (Build 26100). The tests were executed using a single CPU core without GPU acceleration.

A. Metrics

To evaluate the effectiveness of our pruning methods we used the following metrics: Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), Precision (P@10), Recall (Recall@100), and Normalized Discounted Cumulative Gain (NDCG@10). All of these metrics were computed using the official TREC evaluation tool provided by Pyserini. We used appropriate qrel files when needed. In addition to this we also computed the real running time (this time includes all of the warmup steps, which is important for DeepImpact) and also the size of the pruned indices in GB.

B. Comparative Analysis

Our experimental design allows for a comprehensive comparison between:

- BM25 (baseline)
- Full DeepImpact index
- Pruned DeepImpact indices at various retention levels
- Different scoring strategies (document counts, inverse logarithm, inverse bin)

This comparison helps us understand the trade-offs between index size, search efficiency, and retrieval effectiveness as we apply different levels of pruning to the DeepImpact index. By analyzing these results, we aim to identify optimal pruning levels that maintain a balance between index size reduction, query processing speed, and retrieval quality, potentially offering advantages over both the full DeepImpact index and the BM25 baseline in certain scenarios.

V. Results and Discussion

The results for both test datasets are summarized in Table II. These results show the top-1000 documents for each query. For both datasets, we analyzed the size of the indices, real runtime, MAP, MRR, Precision @10, Recall@100, and NDCG @10. The percentage value represents the amount of documents kept from the original index. We included the results for different pruning levels (25%, 50%, and 75%) for count (which are

TABLE II: Search Effectiveness Metrics for Various Indexing Strategies

Model	Size (G)	Time (sec)	MAP	MRR	P@10	Recall@100	NDCG@10
MSMARCO Passage Dev Subset							
BM25	2.6	204	0.1926	0.1960	0.0394	0.6578	0.2284
DeepImpact	1.4	416	0.2994	0.3046	0.0548	0.7827	0.3462
75% Pruned Deep Impact	1.1	298	0.2685	0.2742	0.0480	0.6833	0.3081
50% Pruned Deep Impact	0.69	259	0.2109	0.2168	0.0368	0.5204	0.2403
25% Pruned Deep Impact	0.345	191	0.1302	0.1347	0.0221	0.3047	0.1475
10% Pruned Deep Impact	0.139	140	0.0687	0.0716	0.0109	0.1440	0.0765
5% Pruned Deep Impact	0.070	124	0.0418	0.0435	0.0063	0.0817	0.0458
1% Pruned Deep Impact	0.015	100	0.0133	0.0138	0.0018	0.0210	0.0142
75% Log Pruned Deep Impact	1.1	301	0.2691	0.2747	0.0482	0.6836	0.3088
50% Log Pruned Deep Impact	0.691	237	0.2111	0.2168	0.0368	0.5193	0.2404
25% Log Pruned Deep Impact	0.346	179	0.1319	0.1365	0.0223	0.3049	0.1492
TREC 2020 Test Set							
BM25	2.6	17	0.3440	0.8269	0.5389	0.4834	0.4796
DeepImpact	1.4	87	0.3788	0.8713	0.6537	0.4913	0.6038
75% Pruned Deep Impact	1.1	59	0.3129	0.8446	0.6074	0.4205	0.5572
50% Pruned Deep Impact	0.69	45	0.2181	0.7901	0.5370	0.3035	0.4949
25% Pruned Deep Impact	0.345	34	0.1343	0.7440	0.4241	0.1896	0.3976
10% Pruned Deep Impact	0.139	22	0.0690	0.6177	0.2833	0.0951	0.2809
5% Pruned Deep Impact	0.070	19	0.0332	0.4953	0.1704	0.0470	0.1855
1% Pruned Deep Impact	0.015	18	0.0099	0.1699	0.0296	0.0124	0.0439
75% Log Pruned Deep Impact	1.1	61	0.3146	0.8446	0.6093	0.4239	0.5607
50% Log Pruned Deep Impact	0.691	45	0.2188	0.7832	0.5370	0.3057	0.4959
25% Log Pruned Deep Impact	0.346	31	0.1329	0.7617	0.4204	0.1864	0.3996

referred to as Pruned Deep Impact) and inverse log score schemes; the results for bin score were very similar to log, so we did not include them to keep the tables relatively small. Additionally, we also included pruning levels (1%, 5%, and 10%) for count scores to demonstrate how the system performs under severe pruning.

As you can see, the original DeepImpact index performs better in all metrics compared to BM25, except for the time. We observe a time improvement for DeepImpact indices that keep 25% of the documents or less. For the first dataset, the time clearly does not change proportionally to the size of the index, but it still reduces as we prune. However, we do not see much difference in time for the TREC2020 dataset. This is because the dataset is very small, and the DeepImpact indices require some initialization warm-up step on Pyserini; therefore, we do not observe an improvement in speedup on this dataset. It is worth noting that the time for DeepImpact improves as we evaluate it on larger and larger datasets. For example, the 50% pruned index runs almost 3 times slower compared to BM25 for small dataset, but it runs only 1.26 times slower compared to BM25 using the first dataset. This suggest that in practice, when search is done in large quantities, 50% pruned index can perform on par with BM25 in terms of runtime, while also containing much smaller amount of documents and overall perfoming better in most metrics.

The size of the index also changes roughly proportionally to the pruning level. For example, the ratio of size of 50% pruned index to the original DeepImpact index is 0.49, the ratio of 25% pruned index to the 50% pruned index is exactly 0.5 and so on. This makes sense: assuming documents roughly have the same size and the same number of unique terms, they correspond to roughly the same number of postings. Thus, the size of the indices reduces proportionally to the number of documents removed.

As we prune more, we see a gradual decline in evaluation metrics (MAP, MRR, P@10, NDCG@10), but an interesting observation is that different pruning levels are optimal for different datasets. In the first dataset, we observe that the 25% pruned DeepImpact index still performs better in all metrics compared to the BM25 index; however, the 10% pruned index performs worse. So, the optimal index (in the sense that it is similar to the BM25 index in metrics) should be constructed by keeping somewhere between 10% and 25% of the index. For the second dataset, we observe a different scenario: we observe that the 75% pruned index already performs worse in MAP, and the 50% pruned index performs worse in all metrics but NDCG compared to the BM25 index. So, in this case, pruning does not give us much of an advantage compared to the first dataset.

An interesting application of pruning to consider is heavy pruning (1%, 5%, and 10% levels). These indices obviously cannot achieve the same levels of quality as the original BM25 or DeepImpact indices, but they have a very important advantage in terms of speed. As you can see, for the first dataset, all heavily pruned indices run much faster compared to the BM25 index. At the same time, they can maintain a somewhat moderate level of quality for some metrics. For example, the 10% pruned index still achieves a precision of 0.28 compared to 0.53 for BM25; if we consider the size and running time of a 10% index, we can conclude that this index can find its use in certain scenarios (such as index tiering schemes). However, we have to admit that in many settings, heavily pruned indices do not perform very well and provide poor results.

We would also like to compare the pruning strategies we used. Although our theoretical analysis suggested that we might see a difference in these schemes depending on the distribution of documents in different query results, in practice, we observe very little difference in these values. Coming back to our original motivation for inverse log and inverse bin scores, in practice, it looks like there were not many cases where the system had to decide which document to keep: a document that appears at top-1, top-2, or top-3 results versus a document that appears more often in the results, regardless of its rank. It is likely that in both cases, these documents both had good ranks and appeared often in search.

Overall, the results suggest that in some scenarios we can keep small portions of the index (50% and less) and still perform better than BM25 index. This already gives us an advantage in terms of number of documents we have to store and thus the size of the index. Depending on the application, we may want to keep the dataset smaller, while also performing on par with B25 index. This might be even more important in the case of large, real-life indices, where the number of documents stored is large. At the same time, we could not record any case where we would see an improvement in both quality metrics and time, but as we already discussed, this case is likely for larger datasets.

VI. Limitations and Learnings

While working on this project we faced many difficulties, which we had to overcome. Originally, we used Lucene [Foundation, 2020] as our Information Retrieval (IR) system,

which limited our ability to try out different pruning strategies. Specifically, Lucene's index structures are immutable, which made it difficult for us to modify them and try out techniques like pruning postings lists. We also chose to use PyLucene, a Python wrapper for Lucene as we are more familiar with Python and expected this to give us more flexibility and comparability with other code. However, in fact, this limited our access to some features that are available in the Java version of Lucene, such as the pruning package (org.apache.lucene.index.pruning). We attempted to use this package, but unfortunately, we were unable to get it working with PyLucene.

We also conducted some experiments with PISA (Performant Indexes and Search for Academia) [Mallia et al., 2019], including building and evaluating the indices. Although PISA is a powerful library that provides many useful options for researchers such as different index compression methods and various query processing algorithms, it lacks support for pruning out of the box, and so is not very useful in our scenario. Therefore, we had to proceed with a different library.

Finally, we decided to use Pyserini, a Python library for IR. This allowed us to quickly run experiments and iterate on our approach. However, it also limited our ability to perform custom operations, such as custom topic modeling or tiered searches. These limitations meant that we had to work within the constraints of the Pyserini library, which sometimes restricted our ability to try out new ideas or techniques. If we had more time and resources, we might have considered building a custom IR engine, which would have given us more flexibility to try out different pruning strategies and experiment configurations. This would have allowed us to explore a wider range of techniques and potentially achieve better results.

Overall, we had to spend a lot of time trying different libraries, reading documentation, and exploring different options each library offered until we were able to build something meaningful.

Another issue was the size of the indices and the running time for building and evaluating the indices. Building a single index took about 1.5 hours for 8.8 million collection, and we had to repeat it tens of times while pruning. A single run on train dataset (only 1/4 of the original train dataset) took 4 hours and we also had to do it multiple times. This also slowed down the project development significantly as we could not get the feedback of how our approaches work in a reasonable amount of time.

VII. Future Scope of Work

One simple extension for this project is to attempt to run these pruning schemes on larger query datasets so that we can see an improvement in running time as discussed in the results section. We expect the pruned indices to run much faster relative to BM25 index. If that is true, this pruning scheme might be useful in many scenarios.

Another potential extension of this project could be the development of the new, more advanced pruning strategies. Although we only implemented methods that work by removing documents from the index, we could not find a good way to implement the posting-based or term-based approaches using the indices built by Pyserini. While document-based can be effective in certain scenarios, the posting-based or term-based schemas provide for more flexibity and may achieve much better results. The development of these schemes either using Pyserini or some other library can be a direction of further research. Furthermore, the development of a simple, but convenient framework for static index pruning can be a good project.

Yet another direction of potential research could be the development of theoretical limitations and trade-offs of pruning as well as analysis of optimal pruning strategies. Some papers, such as [Rodriguez and Suel, 2018], were able to achieve some success across this direction, but still, as the authors themselves reported, this still remains an open question.

A. Dynamic Tiering

Another scope of the project that we were working on, but could not complete was to implement dynamic tiering and evaluate results to see if we observe improvement in query time while maintaining the performance of the full Deep Impact index. In our results, we observed that the 50% pruned Deep Impact index was giving good results on the test sets, and we wanted to create two tiers. The first tier is where the query is sent by default (50% pruned Deep Impact index), and the query is redirected to the second tier (full Deep Impact index) only when the top hit from tier 1 match does not satisfy a score cutoff that we set. The logic for that is as follows:

```
import time
import argparse
from pyserini.search import get_topics
from pyserini.search.lucene import LuceneSearcher

# Set up argument parser
parser = argparse.ArgumentParser(description='
    Process queries using tiered search.')
parser.add_argument('--tier1', required=True, help='
    Path to tier 1 index')
parser.add_argument('--cutoff', type=float, required
    =True, help='Score cutoff for tier 1')
parser.add_argument('--tier2', help='Path to tier 2
    index (optional)')
parser.add_argument('--output', default='runs/
    deepimpact_tiered_results.trec', help='Output
    file path')
args = parser.parse_args()

# Initialize searcher for tier 1 index
searcher_tier1 = LuceneSearcher(args.tier1)

# Initialize searcher for tier 2 index if provided
searcher_tier2 = LuceneSearcher(args.tier2) if args.
    tier2 else None

# Get MS MARCO dev subset queries using Pyserini
queries = get_topics('msmarco-passage-dev-subset')

# Function to process queries and write results in
    TREC format
def process_queries(queries, output_file):
    with open(output_file, 'w') as out_f:
        for qid, query in queries.items():
```

```
            query = query['title']
            # Query tier 1
            hits_tier1 = searcher_tier1.search(query, k=1000)

            if hits_tier1 and hits_tier1[0].score >= args.cutoff and searcher_tier2:
                # If top-1 result meets the threshold and tier 2 is available, use tier 1 results
                hits = hits_tier1
                tier = 'tier1'
            elif searcher_tier2:
                # If tier 2 is available, query tier 2
                hits_tier2 = searcher_tier2.search(query, k=1000)
                hits = hits_tier2
                tier = 'tier2'
            else:
                # If tier 2 is not available, use tier 1 results
                hits = hits_tier1
                tier = 'tier1'

            # Write results in TREC format
            for rank, hit in enumerate(hits, start=1):
                out_f.write(f"{qid} Q0 {hit.docid} {rank} {hit.score:.6f} {tier}\n")

# Measure execution time
start_time = time.time()

# Process queries and write results to file
process_queries(queries, args.output)

end_time = time.time()

# Print execution time
print(f"Execution Time: {end_time - start_time:.6f} seconds")
print(f"Results written to {args.output}")
```

TABLE III: Analysis of Qrel Files for TREC 2020 Test Set for Tier Redirection

Relevance Level	Mean	Median	Max	Min	Std. Dev.
Relevance 0	14.1642	13.7729	28.8150	7.6763	3.9987
Relevance 1	15.8509	15.1790	29.8096	9.8160	4.4078
Relevance 2	16.1888	15.7047	31.9263	9.8165	4.8314
Relevance 3	17.4175	15.8845	35.6365	10.3233	5.6382

For finding the cutoff, we conducted analysis on the test set qrel file from TREC 2020, since it provided a broader category of relevance scores (3,2,1,0). According to [Craswell et al., 2020], these scores typically represent different levels of relevance from highly relevant (3) to not relevant (0). A pattern that is visible from the analysis in Table III is that mean deep impact score for documents is increasing as we go higher in the relevance score. This suggests a correlation, and based on this observation, we tried a couple of cutoff scores around the median of relevance 1 and relevance 2 scores, but were unable to test our index performance time due to some bugs we were unable to resolve at this point.

VIII. Conclusion

In this study, we successfully implemented and evaluated pruning schemes for DeepImpact indices, yielding promising results. Our strategies effectively reduced index size while maintaining relatively high evaluation metrics. Notably, we demonstrated that even a 50% pruned DeepImpact index can outperform the BM25 index in certain scenarios, offering a compelling balance between efficiency and effectiveness.

A. Key Findings

- Pruned indices significantly reduced storage requirements, with size decreasing proportionally to the pruning level.
- Query processing time improved for heavily pruned indices, particularly for larger datasets.
- Different pruning levels showed optimal performance across various datasets, suggesting the need for context-specific pruning strategies.
- Heavily pruned indices (1%, 5%, and 10%) demonstrated potential for specialized applications, such as index tiering schemes, despite lower overall performance.

Our results highlight the potential of index pruning as a valuable technique in information retrieval systems, offering a flexible approach to balance index size, query processing speed, and retrieval quality. While we did not observe cases where both quality metrics and processing time improved simultaneously, our findings suggest this may be achievable with larger datasets.

As we move forward, there are numerous avenues for improvement and further research, as outlined in our *Future Scope of Work* section. These developments could further enhance the effectiveness and applicability of pruned indices in real-world information retrieval systems.

REFERENCES

[Anagnostopoulos et al., 2015] Anagnostopoulos, A., Becchetti, L., Bordino, I., Leonardi, S., Mele, I., and Sankowski, P. (2015). Stochastic query covering for fast approximate document retrieval. *ACM Trans. Inf. Syst.*, 33(3).

[Craswell et al., 2020] Craswell, N., Lin, J., Yilmaz, E., Craven, T., Campos, R., Mitra, B., and Trippas, J. (2020). Overview of the trec 2020 deep learning track. In *Proceedings of the 29th Text Retrieval Conference (TREC 2020)*.

[Foundation, 2020] Foundation, A. S. (2020). PyLucene. Accessed: 2020.

[Lassance et al., 2023] Lassance, C., Lupart, S., Déjean, H., Clinchant, S., and Tonellotto, N. (2023). A static pruning study on sparse neural retrievers. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 1771–1775, New York, NY, USA. Association for Computing Machinery.

[Lin et al., 2021] Lin, J., Zhang, X., Ma, X., Wu, W., He, Y., Ye, Z., and Tan, L. (2021). Pyserini: A python toolkit for reproducible search evaluation. https://github.com/castorini/pyserini. Retrieved from: https://github.com/castorini/pyserini.

[Mallia et al., 2021] Mallia, A., Khattab, O., Suel, T., and Tonellotto, N. (2021). Learning passage impacts for inverted indexes. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 1723–1727, New York, NY, USA. Association for Computing Machinery.

[Mallia et al., 2019] Mallia, A., Siedlaczek, M., Mackenzie, J., and Suel, T. (2019). PISA: performant indexes and search for academia. In *Proceedings of the Open-Source IR Replicability Challenge co-located with 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, OSIRRC@SIGIR 2019, Paris, France, July 25, 2019.*, pages 50–56.

[Rodriguez and Suel, 2018] Rodriguez, J. and Suel, T. (2018). Exploring size-speed trade-offs in static index pruning. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1093–1100.