Project 3 Report

Cody Savage
University of Southern Maine
Gorham, Maine, USA

Abstract

In this paper, we will be comparing and analyzing the 'TF-IDF' and 'BM25' retrieval models, and how their performances differ with different preprocessing techniques when given the same information retrieval task, retrieving answers for puzzles and riddles from a given collection of answers to those puzzles and riddles, all retrieved from the internet. In addition, we will evaluate two different neural information retrieval systems and compare the system that is deemed the most effective to the most effective of the traditional models. Finally, we will evaluate two different Large Language Model systems, which use generated text to augment the results of a traditional system.

Keywords: BM25, TF-IDF, PyTerrier, Python, Cross-Encoder, Bi-Encoder, LLM, Transformers

ACM Reference Format:
Cody Savage. 2024. Project 3 Report. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In traditional information retrieval models, the TF-IDF (*term frequency - inverse document frequency*) and Okapi BM25 models are some of the most popular and commonly used.

The TF-IDF model, similar to the simpler Boolean retrieval model, considers documents as *bags of words*, in which terms are weighed independently of each other.

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) * \text{IDF}(t,d)$$

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{TF-IDF}(t,d)$$

Term frequency is measured as the number of times a given term appears in a single document, and inverse document frequency is measured as the log of the total number of documents (N) over the number of documents that the term appears in (n_i). The TF-IDF weight for a given term and document is obtained by multiplying the frequency of the term and the inverse document frequency together, and the rank of a document, given a query q, is the sum of all the TF-IDF weights for terms that appear in both the query and the document [1].

The BM25 model, another *bags-of-words* model, is derived from the binary independence model and is defined as follows:

$$\text{BM25}(t,d) = \text{IDF}(t,d) * \frac{(k+1)tf}{k((1-b)+b*(\frac{L_d}{L_ave}))+tf}$$

$$\text{Score}(q,d) = \sum_{t \in q} \text{BM25}(t,d)$$

Like BIM, BM25 is a probabilistic model, estimating the probability that a given term t appears in a document. BM25 adds to it by introducing a term that takes into account term frequency, document length, and tuning variables k and b, which scale the former and the latter [2].

Both models are well known and regarded, but given how differently they weigh terms, their performance could differ depending on the preprocessing techniques used, even when given the same information retrieval task. Both models take term frequency and and inverse document frequency into account, which could differ depending on how both queries and documents are initially tokenized.

Our next set of models has to do with neural networks. A neural network utilizes a large number of parameters to transform input data, in the form of some sort of vector, into some sort of result. The parameters can be 'tuned' to produce a more desired output. This process of repeated tuning of model parameters is known as *training*, and can be split into three distinct phases:

1. The model's parameters are updated based on a training set of data. Each period of this is known as an *epoch*.
2. The model is validated at the conclusion of each epoch, against a distinct validation set of data, to judge how effective the parameter changes are.
3. After all of the epochs have been completed, the model is evaluated one last time against a test set of data, different from the training and validation sets, to study the model's final efficacy.

Neural models can be used in the field of information retrieval, either to augment existing traditional retrieval systems, or by themselves.

One way neural models are employed are in the form of *Cross-Encoders*. A cross-encoder is a neural model which

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

takes a single sentence pair, combined into a single sequence and separated with a special token, and outputs a value between 0 and 1. This value represents the similarity between the two sentences. By taking in and transforming both sentences at the same time, the model is better at capturing similarity from a semantic perspective versus something like a simple cosine similarity obtained via TF-IDF or BM25 weights, for instance. The downside of a cross-encoder model is that it is very computationally intensive, and not always practical for search through a very large corpus.

Another notable way neural models are utilized are in the form of *Bi-Encoders*. A bi-encoder, instead of taking a pair of sentences, takes only a single sentence, and outputs a final vector 'embedding,' representing the sentence and capturing semantic meaning unavailable to simpler, 'bag-of-word' models. The bi-encoder is specifically trained to produce embeddings that can be compared using cosine similarity, enabling one to rank results given a query. While not as effective as the cross-encoder model, bi-encoders are much faster and efficient in comparison.

It is for this reason that the two models are often combined, using the bi-encoder to generate a smaller selection of results out of a larger corpus, and using the cross-encoder to then *re-rank* said selection of results to obtain a final ranking.

Traditional retrieval systems and Large Language Models can be combined in similar ways. Large Language Models are neural networks, just like Bi-Encoders and Cross-encoders are. However, Large Language Models are consistently trained on much larger sets of data, and are typically based on a *transformer* model, which exclusively use attention-based mechanisms, which are more easily parallelized and perform better when evaluated against other types of models [3]. Large Language Models are often used in text-generation tasks, and can follow specified prompts, which both can be used to enhance information retrieval tasks.

The first technique that we will be examining is called query expansion, where the Large Language Model is prompted to generate text based off a specified query, and the text is appended to the original query in the hopes that the new, modified query returns more relevant results. The idea is that a given user generated query may not perfectly represent the user's information need. For example, a user may not exactly know what they need to know, or they might not use the correct keywords when formulating a query. The goal of modifying the query is to add additional context, so that the underlying system delivers better results. Query expansion can even include up to rewriting the underlying query itself, as it will be seen.

The second technique that we will be examining is one that we discussed prior, that of re-ranking. An initial set of results is obtained via a more efficient system, and after which the Large Language Model is employed to enhanced results in some way, whether it is through query expansion, or via prompting the model to re-rank the documents directly by relevance.

In this paper, both TF-IDF and BM25 models will be compared when given the same set of queries, searching from a collection of answers to puzzles and riddles collected from the *Stack Exchange* website. Using three different preprocessing approaches. Their performances will be compared using standard evaluation metrics, including *P@k* (*precision at k*, measures fraction of relevant results), *nDCG@k* (*normalized discounted cumulative gain at k*, measures how well ranking of results align with their relevance), among other criteria. Afterwards, two different neural retrieval systems will be compared together with the same queries, drawing from the same set of answers. The first model utilizes a single bi-encoder, while the second uses the same bi-encoder in conjunction with a cross-encoder to re-rank the bi-encoder's results. The two different systems will be evaluated using the same metrics as the traditional systems (*P@k*, *nDCG@k*, etc). The most effective systems of the two will be compared to the traditional system deemed most effective. Finally, two systems utilizing Large Language Models, including both query expansion and re-ranking techniques, will be evaluated and compared to the most effective traditional and neural systems.

2 Methodology

The two models used are the two standard models for TF-IDF and BM25 provided by the 'PyTerrier' library (*wmodel='TF_IDF' and wmodel='BM25'*). The three preprocessing techniques are as described and labeled as follows:

1. **Basic:** Default PyTerrier stopword removal and stemming; Punctuation is removed and tokens are lowercased.
2. **HTML:** Text is split between a HTML tag regex pattern and then joined together. Very small substrings are ignored.
3. **Advanced:** Mathematical symbols like '+' or '-' are replaced with expanded tokens like 'plus' or 'minus.'

In addition, more detailed preprocessing techniques perform the same steps as the less detailed techniques below them (e.g. 'Advanced' preprocessing also includes 'Basic' and 'HTML' preprocessing, and so forth).

For the neural models, the six different permutations of models are tested on three different sets of the same queries, each set of queries also being reprocessed with the same techniques above. Each query corresponds to 100 results. Mean and per-query measures were collected, resulting in 18 sets of mean metrics collected, and 18 sets of per-query metrics collected, every set of metrics in each set corresponding to a model plus it's preprocessing technique, and a query set plus it's preprocessing technique.

The bi-encoder uses the *sentence-transformers/multi-qa-mpnet-base-cos-v1* model, which was trained on 215 million

question-answer pairs from websites like *StackExchange*, *Google*, *Bing*, among others. The cross-encoder uses the *cross-encoder/ms-marco-MiniLM-L-4-v2* model, trained on the *MS MARCO* passage ranking task, and well suited for general purpose re-ranking. Both models are used with the *sentence-transformers* Python library. All sets of queries and the answers were preprocessed with HTML preprocessing, albeit with the punctuation included instead of being removed.

The two systems, bi-encoder only and with cross-encoder re-ranking, are tested on the same set of the queries, each query corresponding to 100 results. Mean and per-query measures were collected, resulting in 2 sets of mean metrics collected, and 2 sets of per-query metrics collected. Given the limited data set that may be obtained from the set of queries and answers, and the recorded performance of the pre-trained models, the decision was made to not fine-tune the models. For measuring efficiency, time to create the embeddings for the answer collection and the respective set of queries is included. Re-ranker times also include the time elapsed for the bi-encoder run behind it. Also to note is that time measures for both neural systems also includes the embedding encoding time of the collection and the queries used.

For the systems which utilize the Large Language Models, the single, underlying model used is the *Qwen/Qwen2.5-3B-Instruct*, which is used with the *transformers* Python library. The model was picked due to it's performance in relation to it's size (3 billion parameters), maximizing it's efficiency. In addition, its model card on *HuggingFace* states that *Qwen2.5* has increased performance when it comes to mathematics, which is a desired attribute considering the dataset that we have. Broadly speaking, two systems, using the same model, are:

1. **Query-Expansion:** All queries are directly augmented/re-written, before being processed by a traditional retrieval system.
2. **Re-ranking:** The top-100 results are obtained with the original queries and a traditional retrieval system. Then, query expansion is performed on the queries, and the top-100 results are re-ranked by the traditional system with the new queries.

The underlying traditional system used is the BM25 model as provided by the *PyTerrier* library. For Each system, two prompts and respective query expansion approaches are used:

1. **Appending (Prompt 1):**
   a. *Prompt:* 'Given the following query, provide a comma separated list of keywords that are relevant to the query. Provide at least ten keywords.'
   b. *Approach:* The generated list of keywords is appended to the original query.
2. **Rewriting (Prompt 2):**
   a. *Prompt:* 'Given the following query, answer it with a single paragraph.'
   b. *Approach:* The original query is replaced with the generated paragraph.

Both prompts are *zero-shot* prompts: zero examples are provided to the Large Language Model, and the model is expected to answer the prompt directly. Other types of prompting include *few-shot* and *chain-of-thought* prompting, but are outside the scope of this paper. Each system was ran on both prompts, on both sets of queries, resulting in eight sets of results. All sets of queries and the answers were preprocessed with HTML preprocessing, minus the generated text, and since the underlying system is provided by *PyTerrier*, punctuation is removed from the queries before each query. All measures were collected with the *trec_eval* tool.

Name	**Basic TFIDF**	**Basic BM25**
P.5	0.4577	0.4593
P.10	0.3011	0.3038
P.100	0.0398	0.04
ndcg_cut.5	0.4561	0.4569
ndcg_cut.10	0.4747	0.4778
ndcg_cut.100	0.5211	0.5237
map	0.442	0.4452
recip_rank	0.6836	0.6843
Name	**HTML TFIDF**	**HTML BM25**
P.5	0.4724	0.4783
P.10	0.3134	0.318
P.100	0.0415	0.0419
ndcg_cut.5	0.4699	0.4746
ndcg_cut.10	0.4912	0.497
ndcg_cut.100	0.5397	0.5448
map	0.4602	0.4659
recip_rank	0.6956	0.6975
Name	**Adv. TFIDF**	**Adv. BM25**
P.5	0.4664	0.471
P.10	0.3093	0.3134
P.100	0.0413	0.0417
ndcg_cut.5	0.4652	0.4694
ndcg_cut.10	0.4871	0.4919
ndcg_cut.100	0.5358	0.5408
map	0.4544	0.4597
recip_rank	0.6901	0.6918

Table 1. All Models - Basic Queries

3 Experimental Analysis

Efficiency-wise, the average time per model was very similar, imaginably because the underlying models are the same, with only the preprocessing techniques differing. The advanced set of queries consistently took slightly longer, likely

Name	Basic TFIDF	Basic BM25
P.5	0.4657	0.4705
P.10	0.3079	0.3116
P.100	0.0403	0.0408
ndcg_cut.5	0.4622	0.4646
ndcg_cut.10	0.4833	0.4869
ndcg_cut.100	0.5286	0.5331
map	0.4509	0.4554
recip_rank	0.6888	0.6902
Name	**HTML TFIDF**	**HTML BM25**
P.5	0.4868	0.4916
P.10	0.3218	0.326
P.100	0.0422	0.0426
ndcg_cut.5	0.4816	0.4852
ndcg_cut.10	0.503	0.5078
ndcg_cut.100	0.5509	0.556
map	0.4742	0.4801
recip_rank	0.711	0.7127
Name	**Adv. TFIDF**	**Adv. BM25**
P.5	0.4795	0.4836
P.10	0.3193	0.3218
P.100	0.042	0.0425
ndcg_cut.5	0.4782	0.482
ndcg_cut.10	0.5008	0.5046
ndcg_cut.100	0.5484	0.5538
map	0.4685	0.4738
recip_rank	0.705	0.7064

Table 2. All Models - HTML Queries

Name	Basic TFIDF	Basic BM25
P.5	0.3816	0.3965
P.10	0.2476	0.2561
P.100	0.0335	0.0348
ndcg_cut.5	0.3837	0.3979
ndcg_cut.10	0.3968	0.4105
ndcg_cut.100	0.4378	0.4538
map	0.361	0.3754
recip_rank	0.588	0.6044
Name	**HTML TFIDF**	**HTML BM25**
P.5	0.4152	0.4294
P.10	0.2707	0.2799
P.100	0.0359	0.0369
ndcg_cut.5	0.4148	0.4278
ndcg_cut.10	0.4293	0.4427
ndcg_cut.100	0.4703	0.484
map	0.3961	0.4101
recip_rank	0.6178	0.6334
Name	**Adv. TFIDF**	**Adv. BM25**
P.5	0.4987	0.485
P.10	0.3325	0.322
P.100	0.0425	0.0421
ndcg_cut.5	0.498	0.4814
ndcg_cut.10	0.5226	0.5042
ndcg_cut.100	0.5652	0.5505
map	0.4887	0.4729
recip_rank	0.7248	0.7044

Table 3. All Models - Adv. Queries

because of the inserted tokens in place of normal mathematical symbols.

Overall, it is seen that the queries that used the same preprocessing techniques as the model either did about as well, or better than if their preprocessing techniques didn't match. There isn't a strong correlation among the basic query results, but the HTML models performed strongest with the HTML query set, with P@5s and P@10s highest in it's set, and higher than either of their performances in the other sets.

Likewise, the advanced models performed strongest with the advanced queries, with some of the best P@5s and P@10s out of all the sets, regardless of the set of queries being used.

An example of a query which benefited from the additional preprocessing would be the query with the ID '125245'. With basic preprocessing and corresponding model, the query had a P@5 of 0 across both TF-IDF and BM25 models. With HTML preprocessing, this increases to 0.2, and 0.6 with advanced preprocessing. The query itself has many HTML tags littered in the query body, including an web link, which likely degenerate to irrelevant terms after basic preprocessing, which might lead it to falsely be deemed relevant with other documents which contain said irrelevant terms. The HTML preprocessing eliminates the irrelevant terms, while the mathematical symbol replacement replaces remaining symbols, most notably periods with the term ' dot ', which evidently increases matching relevant terms.

Experiment	Ave. Time / Model	Total Time
Basic Queries	102.44s	10m, 14s
HTML Queries	102.76s	10m, 16s
Adv. Queries	110.49s	11m, 02s

Table 4. Sampled Time Elapsed From Experiment Run

This example helps highlights one of the greatest strengths of the added preprocessing, which appears to be the HTML tag elimination. This lines up with the assumption that, since the data is retrieved from a website in relatively unmodified form, the additional, irrelevant text as a result hampers IR models without additional consideration.

However, seeing how the symbol replacement may effect a query also brings to mind a potential weakness. It is possible that symbols might get replaced with the new mathematical-related terms, even when the piece of text doesn't have to

Name	Bi-Encoder	Re-ranker	Adv. TF-IDF
P.5	0.5018	0.4329	0.4987
P.10	0.3394	0.2992	0.3325
P.100	0.0471	0.0471	0.0425
ndcg_cut.5	0.4999	0.4272	0.4980
ndcg_cut.10	0.5271	0.4569	0.5226
ndcg_cut.100	0.5914	0.5435	0.5652
map	0.4910	0.4316	0.4887
recip_rank	0.7497	0.6634	0.7248

Table 5. Measure of Neural Systems and most Effective Traditional System (Adv. TF-IDF)

Queries	Time Elapsed
Topic 1	9m, 12s
Topic 2	9m, 7s

Table 6. Measure of time elapsed per query set for the Bi-Encoder System

do with mathematics. If this happens to a document, then the document may appear more relevant to mathematically-related queries than it really is. Likewise, it might make a given query appear more relevant to mathematically-related documents as well.

An improvement to the advanced preprocessing in the future could be to selectively replace symbols only if it can be determined if the piece of text is mathematically related, and to not and simply just strip away said symbols otherwise. For example, this could be determined through associated metadata, like tags, or in it's absence, some sort of heuristic based on the text's existing terms.

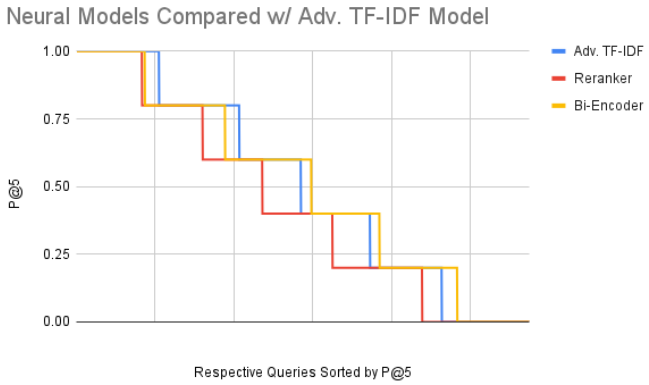

Figure 1. P@5 per query compared between Neural Systems and Adv. TF-IDF

We will now discuss the results of the neural systems.

The measure for efficiency, or time spent, for each system and set of queries is not surprising, and equally not surprising is that it doesn't compare very well to the times of the traditional systems. The traditional systems are much simpler than their neural counterparts, even when counting some of the more advance preprocessing techniques used. The second query set was marginally quicker than the the first set for both systems (see tables 6 and 7), which follows expectations, since the second set has 436 fewer queries to process.

Queries	Time Elapsed
Topic 1	26m, 6s
Topic 2	23m, 38s

Table 7. Measure of time elapsed per query set for the Re-ranker System

Overall, both models performed fairly comparably, with the system solely using the bi-encoder performing slightly better than the re-ranking system (see table 5). Disappointingly, the system that used the re-ranking performed noticeably worse than both the bi-encoder system and many of the traditional systems, with lower metrics across the board.

While the metrics for the bi-encoder system are highest compared to all other systems evaluated thus far, by graphing the precision at 5 vs queries and comparing said graph to the best of the traditional systems, it can be seen that there are more queries with a P@5 > 0.6 from the simpler, traditional system versus the bi-encoder (see figure 1).

An example of a query getting hindered by bi-encoder would be the one with the id *2147*, which asks, 'help understanding the solution to a "shooting puzzle"', and more specifically, speaks of a mathematical puzzle involving probability. The queries corresponding results obtained a P@5 of 0.4 and 0.8 from the bi-encoder and re-ranking systems respectively, but the results from the Adv. TF-IDF system obtained a 'perfect' P@5, a value of 1.

Another example of a query performing sub-optimally with the neural systems would be the query with id *6066*, asking, 'can you solve those laser puzzles?', a question that specifically has to do with geometry, and by extension, more mathematics. The question also presents a image, which presumably has to do with the lasers in question. Again, the Adv. TF-IDF system presents a perfect P@5 of 1, while the two neural systems present with the direct opposite, the worst possible value of P@5, which is 0.

The general trend suggested by the figure 1 is that the bi-encoder system trades optimal precision for some queries to have an improved precision for queries that performed sub-optimally with the tradition system. Given the nature of these neural systems, it can be hypothesized why this is the case. by virtue of being supported by machine learning models, can encode semantical information effectively out of reach of a traditional system. However, our neural systems are trained on large, diverse datasets, and it therefore makes

Name	Query Ex.	Re-ranker
P.5	0.5140	0.5151
P.10	0.3433	0.3436
P.100	0.0444	0.0445
ndcg_cut.5	0.5092	0.5107
ndcg_cut.10	0.5345	0.5357
ndcg_cut.100	0.5817	0.5832
map	0.5048	0.5064
recip_rank	0.7283	0.7298

Table 8. Measure of Large Language Model Systems (Prompt 1)

Name	Query Ex.	Re-ranker
P.5	0.3547	0.3624
P.10	0.2422	0.2480
P.100	0.0373	0.0375
ndcg_cut.5	0.3605	0.3664
ndcg_cut.10	0.3832	0.3893
ndcg_cut.100	0.4490	0.4524
map	0.3463	0.3516
recip_rank	0.5652	0.5792

Table 9. Measure of Large Language Model Systems (Prompt 2)

sense why, when handed more specific or narrow-focused queries, they may falter in ways that a traditional model doesn't. A neural model is limited by both it's complexity, and more notably, the dataset it has been trained on, which could even limit said model in certain domains, even more so than how a traditional information retrieval system like TF-IDF or BM25 is by nature.

In short, for the 'average' cases, the neural model can, and apparently does, perform better, but in more 'niche' cases, it may evidently perform worse.

This is also why I believe that the re-ranking system did particularly worse, compared to all else thus far. The cross-encoder which re-ranks the bi-encoder's results was trained on a different dataset than the bi-encoder, further widening the total scope of data that the system has been conditioned to, and further limiting performance when it comes to more specific topics.

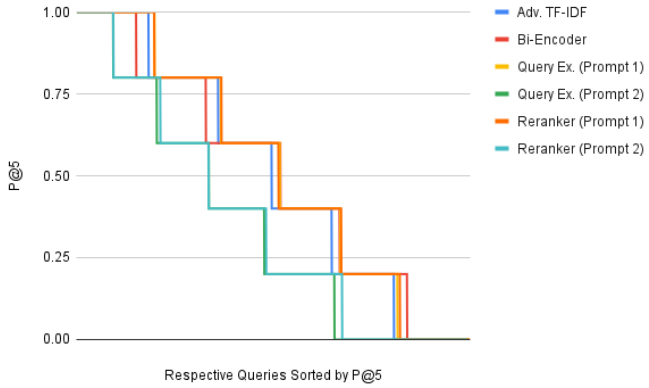

Figure 2. P@5 per query compared between LLM Systems, Bi-Encoder, and Adv. TF-IDF

Now, we will discuss the systems which used the Large Language Model.

To touch upon efficiency, both systems took a significant amount of time to run, compared to both the traditional and neural systems. This perfectly lines up with expectations,

Name	Re-ranker (Prompt 1)	Adv. TF-IDF
P.5	0.5151	0.4987
P.10	0.3436	0.3325
P.100	0.0445	0.0425
ndcg_cut.5	0.5107	0.4980
ndcg_cut.10	0.5357	0.5226
ndcg_cut.100	0.5832	0.5652
map	0.5064	0.4887
recip_rank	0.7298	0.7248

Table 10. Measures of best Large Language Model system with best Traditional System

Name	Re-ranker (Prompt 1)	Bi-Encoder
P.5	0.5151	0.5018
P.10	0.3436	0.3394
P.100	0.0445	0.0471
ndcg_cut.5	0.5107	0.4999
ndcg_cut.10	0.5357	0.5271
ndcg_cut.100	0.5832	0.5914
map	0.5064	0.4910
recip_rank	0.7298	0.7497

Table 11. Measures of best Large Language Model system with best Neural System

as since the neural systems took much longer than the traditional systems, and the Large Language Model systems are, as the name suggests, *larger* neural network models, so it makes sense that they would be the longest to run. The re-ranking systems took less time to run than the query expansion systems, which also follows expectations. The re-ranking systems only invoke the Large Language Model for the 100 queries returned from the initial query, instead on all of the queries at once, which is faster. It is worth noting that the re-ranker does invoke the traditional system twice, versus just once for the query expansion systems, but the Large Language Model dominates the runtime, so this isn't noticeable.

For each system, when utilizing the same prompting strategy, they perform very similarly. This can be seen, for example, when comparing the nDCG@100 values for the two systems when using prompt 1 (see table 8). In essence, nDCG measures how close two systems are to an 'optimal' ranking, where a score of 1 at rank k indicating the system has ranked the results perfectly up to rank k. Both systems, query expansion and re-ranker, have nDCG@100 scores of 0.5817 and 0.5832 respectively. The difference between the two is a mere 0.0015, indicating the final ranking of both systems is very similar to each other. This can also be seen to a lesser extent when taking the second prompting strategy into account (see table 9). The nDCG@100 score of the query expansion system is 0.4490, and the nDCG@100 score of the re-ranking system is 0.4524. There is more of a difference, but the two score are still very close, everything else considered.

I hypothesize that the evident similarities of the two systems is likely a result of both using the same, underlying traditional model for all the actual retrieval tasks. Recall that said model, BM25, is a simple 'bag-of-words' model, and most importantly, compared to something like a system using cross-encoders for example, is quite deterministic, so unless the underlying query is drastically modified, it appears that in what order or how one uses the underlying model is of less importance. This can also be reflected in the difference in performance between the two difference prompts, where the query is modified in very different ways, and indeed, we do get very different results in the end (and not for the better, either).

The systems which used the first prompting strategy versus the second performed significantly better in almost every single aspect (see tables 8 and 9). For example, considering the re-ranking system, using prompt 1 granted a P@5 of 0.5151, while the exact same system using prompt 2 resulted in a P@5 of 0.3624, a value about 70% of the original.

Again, I believe this may be because of the traditional retrieval model that both systems, no matter the prompt rely on. Again, BM25 considers each word independently, with no consideration to the semantics or surrounding words. The second prompting strategy entirely replaces the original query with a generated 'answer,' which is used instead, while the first merely appends a list of keywords to the existing query. While the generated 'answer' may match the relevant results semantically, it doesn't appear that they matched very well in any other aspect, as the final measures suggest. A list of keywords may not be very coherent in a semantic sense, but a diverse and relevant set of keywords is exactly what BM25 is looking for, and keeping the original context in the form of not discarding the original query, and thus keeping the words most likely referenced in the original corresponding questions when the query was first made likely helps as well.

An example of a query which benefited from this original context is the query with the id *4247*, which asks 'are you clever enough?' The actual content of the query is a riddle, which states that the answer 'is a single, short word.' With the first prompt, the top answers from both the query expansion and re-ranking systems is the answer with id *4290*, which suspects that the answer might have something to do with Sherlock Homes, and paraphrases some of what the original query said. Meanwhile, when looking at the results from the second prompt, both system returned answer with id *61498* as the top result. The answer in question asks if the person asking is referring to what they call a 'unsolvable riddle,' which I don't think matches the original riddle very well, except for the face that it is a riddle, which doesn't narrow things down very well. The raw measures support my line of thinking, as both systems obtained a P@5 of 0.6 with the query while using the first prompt, and a much lower P@5 of 0 with the second.

As an interesting counter-example, there exists a query with the id of *22812*, which is titled 'Famous Last Words (or Escape the Concatenation Camp).' The body of the query describes a somewhat convoluted logical puzzle, where you have to figure out how to save a group of words from the aforementioned camp. The top answers from both systems using prompt 1, answer with id *49386*, lists thirteen words that don't appear to be relevant to the original query. Now looking at the results from the query expansion, the re-ranker, and prompt 2, they returned *23568* and *22881* respectively. Both answers specifically mention the original group of words from the query, and are clearly relevant to the query. As these answers imply, both systems with the first prompt got a P@5 of 0, while they got a P@5 of 1 with the second, for this query. While a select few queries performed much better with the second prompt, this is much the exception rather than the norm.

In summary, the system that performed the best out of the other combination of Large Language Model systems and prompts was the re-ranker system with the first prompt, performing slightly better than it's counterpart. In comparison with the best systems from the neural models and the traditional models, those being the Bi-Encoder and the Adv. TF-IDF, the first prompt re-ranker beats out both in being the best evaluated retrieval system out of every single one compared. Of note is the same phenomenon which occurred between the Adv. TF-IDF and Bi-Encoder repeating between the Bi-Encoder and the re-ranker. As previously stated, it appears that the Bi-Encoder trades lower initial precision for a slightly higher overall precision, and the same can both be seen in the graph (see figure 2) and in their respective nDCG@100 values (see table 11). While the re-ranker initially has higher nDCG values for low k, the Bi-Encoder actually has a higher nDCG for all (100) results, which implies that it's overall ranking is slightly better. This calls into question over which is the 'best' overall system, which if we were to split the tie with efficiency, the Bi-encoder appears to be the most effective.

4 Conclusion

After using more sophisticated preprocessing techniques, evidence suggests that the performance increases, assuming both the originally-indexed documents and incoming queries are processed the same way. This suggests that different and improved preprocessing techniques can help improve and augment existing information retrieval models, and can even be fine-tuned for specific use-cases and data to further refine models in the future.

The results from our neural systems suggest with much certainty that neural systems, in general purpose information retrieval tasks, can out perform traditional information retrieval systems like TF-IDF or BM25. However, as it can also be seen, the performance of neural systems also lean heavily not just on their complexity, but the breadth and content of the different datasets the models which power them are trained on. This suggests that for more purpose-driven tasks, such as our task for retrieving answers to puzzles and riddles, a model with an equally purpose-driven dataset and training is required to maximize performance. This is especially important when taking in mind the vast differences in computational complexity and efficiency that divides traditional and neural models. Perhaps, when taking into account the increase of computational complexity that a domain-specific neural model entails, including the additional long-term increase of power usage and cost, that traditional systems, either by themselves or in conjunction with less sophisticated neural models, may be worth exploring in the future.

Finally, we look at our results from the Large Language Model systems. The results do hold much promise, but the evidence suggests that their performance depends strongly on how they're used, what prompts are used, and what underlying systems are used in conjunction with them. If the wrong combination of prompt and system is used, then it appears that it can severely limit their performance. Like with the neural systems, the notion of efficiency also comes into question, and even more so considering the increase of complexity of a Large Language Model versus a neural model. This too suggests that the most effective final system balances both efficiency and performance, perhaps with a combination of the above systems.

References

[1] Stephen Robertson. 2004. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of documentation* 60, 5 (2004), 503–520.

[2] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. 2004. Simple BM25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. 42–49.

[3] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

Received 9 December 2024