# Adaptive Re-Ranking with a Corpus Graph

Sean MacAvaney
sean.macavaney@glasgow.ac.uk
University of Glasgow
Glasgow, United Kingdom

Nicola Tonellotto
nicola.tonellotto@unipi.it
University of Pisa
Pisa, Italy

Craig Macdonald
craig.macdonald@glasgow.ac.uk
University of Glasgow
Glasgow, United Kingdom

## ABSTRACT

Search systems often employ a re-ranking pipeline, wherein documents (or passages) from an initial pool of candidates are assigned new ranking scores. The process enables the use of highly-effective but expensive scoring functions that are not suitable for use directly in structures like inverted indices or approximate nearest neighbour indices. However, re-ranking pipelines are inherently limited by the recall of the initial candidate pool; documents that are not identified as candidates for re-ranking by the initial retrieval function cannot be identified. We propose a novel approach for overcoming the recall limitation based on the well-established clustering hypothesis. Throughout the re-ranking process, our approach adds documents to the pool that are most similar to the highest-scoring documents up to that point. This feedback process *adapts* the pool of candidates to those that may also yield high ranking scores, even if they were not present in the initial pool. It can also increase the score of documents that appear deeper in the pool that would have otherwise been skipped due to a limited re-ranking budget. We find that our *Graph-based Adaptive Re-ranking* (Gar) approach significantly improves the performance of re-ranking pipelines in terms of precision- and recall-oriented measures, is complementary to a variety of existing techniques (e.g., dense retrieval), is robust to its hyperparameters, and contributes minimally to computational and storage costs. For instance, on the MS MARCO passage ranking dataset, Gar can improve the nDCG of a BM25 candidate pool by up to 8% when applying a monoT5 ranker.[1]

## CCS CONCEPTS

• **Information systems → Retrieval models and ranking**.

## KEYWORDS

neural re-ranking, clustering hypothesis

**ACM Reference Format:**
Sean MacAvaney, Nicola Tonellotto, and Craig Macdonald. 2022. Adaptive Re-Ranking with a Corpus Graph. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557231

[1]Code for this work is available at https://github.com/terrierteam/pyterrier_adaptive.

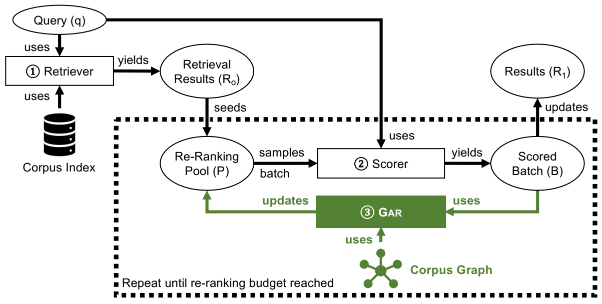

**Figure 1: Overview of Gar. Traditional re-ranking exclusively scores results seeded by the retriever. Gar (in green) adapts the re-ranking pool after each batch based on the computed scores and a pre-computed graph of the corpus.**

## 1 INTRODUCTION

Deep neural ranking models – especially those that use contextualised language models like BERT [6] – have brought significant benefits in retrieval effectiveness across a range of tasks [18]. The most effective techniques tend to be those that first retrieve a pool of candidate documents[2] using an inexpensive retrieval approach and then re-score them using a more expensive function. This process is called *re-ranking*, since the documents from the candidate pool are given a new ranked order. Re-ranking enables the use of sophisticated scoring functions (such as cross-encoders, which jointly model the texts of the query and document) that are incompatible with inverted indexes or vector indexes. Since the scoring function can be computationally expensive, re-ranking is often limited to a predefined maximum number documents that the system is willing to re-rank for each query (i.e., a *re-ranking budget*, such as 100).

The performance of a re-ranking pipeline is limited by the recall of the candidate pool, however. This is because documents that were not found by the initial ranking function have no chance of being re-ranked. Consequently, a variety of techniques are employed to improve the recall of the initial ranking pool, including document-rewriting approaches that add semantically-similar terms to an inverted index [30], or dense document retrieval techniques that enable semantic searching [12].

In this work, we explore a complementary approach to overcoming the recall limitation of re-ranking based on the long-standing *clustering hypothesis* [11], which suggests that closely-related documents tend to be relevant to the same queries. During the re-ranking process, our approach, called *Graph-based Adaptive Re-Ranking* (Gar), prioritises the scoring of the neighbours of documents that have received high scores up to this point. An overview of Gar is shown in Figure 1. The Gar feedback mechanism allows for

[2]Or passages; we often simply use the term "document" for ease of reading.

documents to be retrieved that were not present in the initial re-ranking pool, which can improve system recall. It also allows for the re-ranking of the documents that may have otherwise been skipped from the pool when the re-ranking budget is low. Finally, by including the feedback within the re-ranking itself (as opposed to post-scoring feedback mechanisms, such as PRF), our approach can find documents that are multiple hops away (i.e., neighbours of neighbours). Gar achieves low online overhead through offline computation of a *corpus graph* that stores the nearest neighbours of each document.

On experiments over the TREC Deep Learning datasets, we find that Gar significantly improves precision- and recall-oriented evaluation measures. Gar can improve virtually any re-ranking pipeline, with the results largely holding across a variety of initial retrieval functions (lexical, dense retrieval, document expansion, and learned lexical), scoring functions (cross-encoding, late interaction), document similarity metrics (lexical, semantic), and re-ranking budgets (high, low). Impressively, a Gar pipeline that uses only BM25 for both the initial retrieval and the document similarity is able to achieve comparable or improved performance in terms of re-ranked precision and recall over the competitive TCT-ColBERT-v2-HNP [19] and DocT5Query [30] models – both of which have far higher requirements in terms of offline computation and/or storage capacities. We find that the online overhead of Gar is low compared to a typical re-ranking, usually only adding around 2-4ms per 100 documents re-ranked. We also find that Gar is largely robust to its parameters, with major deviations in performance only occurring with extreme parameter values. Finally, we find that despite using document similarity, Gar does not significantly reduce the diversity among the relevant retrieved documents.

In summary, we propose a novel approach to embed a feedback loop within the neural re-ranking process to help identify un-retrieved relevant documents through application of the clustering hypothesis. Our contributions can therefore be summarised as follows: (1) We demonstrate a novel application of the clustering hypothesis in the context of neutral re-ranking; (2) We show that our proposed approach can successfully improve both the precision and the recall of re-ranking pipelines with minimal computational overhead; (3) We demonstrate that the approach is robust across pipeline components and the parameters it introduces. The remainder of the paper is organised as follows: We first provide additional background and related work, positioning Gar in context with past work in neural retrieval, relevance feedback, and the clustering hypothesis (Section 2); We then briefly demonstrate that the clustering hypothesis still holds on a recent dataset to motivate our approach (Section 3); We formally describe our method (Section 4) and present our experiments that demonstrate its effectiveness (Sections 5 & 6); We wrap up with final conclusions and future directions of this promising area (Section 7).

## 2 BACKGROUND AND RELATED WORK

The recent advancements in deep neural ranking models have brought significant improvements on the effectiveness of ad-hoc ranking tasks in IR system [18]. In particular, pre-trained language models such as BERT [6] and T5 [33] are able to lean semantic representations of words depending on their context, and these representations are able to better model the relevance of a document w.r.t. a query, with notable improvements w.r.t. classical approaches. However, these improvements have an high computational costs; BERT-based rankers [22, 28] are reported to be slower than classical rankers such as those based on BM25 by orders of magnitude [10, 22]. Therefore, it is still usually infeasible to directly use pre-trained language models to rank all documents in a corpus for each query (even using various to reduce the computational cost [12, 20, 21].) Deep neural ranking models are typically deployed as re-rankers in a pipeline architecture, where a first preliminary ranking stage is deployed before the more expensive neural re-ranker, in a cascading manner. During query processing, the first ranking stage retrieves from the whole document corpus a candidate pool of documents using a simple ranking function, with the goal of maximising the recall effectiveness. The following re-ranking stage processes the documents in the candidate pool, reordering them by focusing on high precision results at the top positions, whose documents will be returned to the user [31, 36]. In this setting, there is an efficiency-effectiveness tradeoff on the number of documents retrieved by the first ranker. From the efficiency perspective, a smaller number of documents in the candidate pool will allow the re-ranker to reduce the time spent on re-ranking the documents, since the execution time is proportional to the candidate set size. From the effectiveness perspective, the larger the candidate pool, the higher the number of potentially relevant documents to be retrieved from the document corpus. In fact, relevant documents can be retrieved from the corpus only during first-stage processing. The recall effectiveness of the candidate pool has been investigated in previous IR settings, in particular in learning-to-rank pipelines. Tonellotto et al. [35] studied how, given a time budget, dynamic pruning strategies [36] can be use in first-stage retrieval to improve the candidate pool size on a per-query basis. Macdonald et al. [23] studied the minimum effective size of the document pool, i.e., when to stop ranking in the first stage, and concluded that the smallest effective pool for a given query depends, among others, on the type of the information need and the document representation. In the context of neural IR, learned sparse retrieval focuses on learning new terms to be included in a document before indexing, and the impact scores to be stored in the inverted index, such that the resulting ranking function approximates the effectiveness of a full transformer-based ranker while retaining the efficiency of the fastest inverted-index based methods [5, 7, 26]. In doing so, first-stage rankers based on learned impacts are able to improve the recall w.r.t. BM25, but the end-to-end recall is still limited by the first-stage ranker.

Pseudo-Relevance Feedback (PRF) involves the reformulation of a query based on the top results (e.g., by adding distinctive terms from the top documents). This query is then re-issued to the engine, producing a new ranked result list. Adaptive Re-Ranking also makes use of these top-scoring documents, but differs in two important ways. First, the query remains unmodified, and therefore, ranking scores from the model need not be re-computed. Second, the top scores are used in an intermediate stage of the scoring process; the process is guided by the highest-scoring documents known up until a given point, which may not reflect the overall top results. Finally, we note that the output of an adaptive re-ranking operation could be fed as input into a PRF operation to perform query reformulation.

This work can be seen as a modern instantiation of the clustering hypothesis, which Jardine and van Rijsbergen [11] stated as *"Closely associated documents tend to be relevant to the same requests"*. Many works have explored the clustering hypothesis for various tasks in information retrieval, such as for visualisation of the corpus (e.g., [17]), visualisation of search results (e.g., [4]), enriching document representations [16] and fusing rankings (e.g., [15]). Most related to our application is the usage of the clustering hypothesis for first-stage retrieval (i.e., document selection), in which the documents to rank are identified by finding the most suitable cluster for a query [13]. However, these works focus on identifying the most suitable clusters for a given query and transforming the constituents into a ranking. Moreover, while our approach also takes a soft clustering approach [14] where each 'cluster' is represented by a document and its neighbours, instead of ranking clusters, we identify "good" clusters as when the representative document is scored highly by a strong neural scoring function. We also address the problem of transforming the documents into a ranking by letting the neural scoring function do that job as well. Overall, our novel approach is the first to embed a feedback loop within the re-ranking process to help identify un-retrieved relevant documents.

## 3 PRELIMINARY ANALYSIS

We first perform a preliminary check to see whether the clustering hypothesis appears to hold on a recent dataset and using a recent model. Namely, we want to check whether the passages from the MS MARCO corpus [2] are more likely to distributed closer to those with the same relevance labels than those with differing grades. We explore two techniques for measuring similarity: a lexical similarity score via BM25, and a semantic similarity via TCT-ColBERT-HNP [19]. For the queries in the TREC DL 2019 dataset [3], we compute similarity scores between each pair of judged documents. Then, akin to the Voorhees' cluster hypothesis test [37], we calculate the distribution of the relevance labels of the nearest neighbouring passage by relevance label (i.e., we calculate $P\big(rel(neighbour(p)) = y | rel(p) = x\big)$ for all pairs of relevance labels $x$ and $y$.)

Table 1 presents the results of this analysis. We observe a clear trend: passages with a given relevance label are far more likely to be closer to the passages with the same label (among judged passages) than those with other labels (in the same row). This holds across both lexical (BM25) and semantic (TCT-ColBERT) similarity measures, and across all four relevance labels (ranging from non-relevant to perfectly relevant).

**Table 1: Distribution of nearest neighbouring passages, among pairs of judged passages in TREC DL 2019, based on BM25 and TCT-ColBERT-HNP similarity scores. Each cell represents the percentage that a passage with a given relevance label ($x$) has a nearest neighbour with the column's relevance label ($y$); each row sums to 100%.**

| | BM25 | | | | TCT-ColBERT-HNP | | | |
|---|---|---|---|---|---|---|---|---|
| | neighbour's rel $y$ | | | | neighbour's rel $y$ | | | |
| | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| $x = 0$ | **67** | 11 | 16 | 7 | **76** | 10 | 10 | 4 |
| 1 | 14 | **47** | 31 | 8 | 17 | **46** | 29 | 8 |
| 2 | 8 | 12 | **71** | 9 | 8 | 11 | **72** | 8 |
| 3 | 8 | 7 | 12 | **73** | 6 | 7 | 12 | **75** |

**Algorithm 1** Graph-based Adaptive Re-Ranking

```
Input: Initial ranking $R_0$, batch size $b$, budget $c$, corpus graph $G$
Output: Re-Ranked pool $R_1$
  $R_1 \leftarrow \emptyset$                                    ▷ Re-Ranking results
  $P \leftarrow R_0$                                    ▷ Re-ranking pool
  $F \leftarrow \emptyset$                                    ▷ Graph frontier
  do
      $B \leftarrow$ Score(top $b$ from $P$, subject to $c$)         ▷ e.g., monoT5
      $R_1 \leftarrow R_1 \cup B$                               ▷ Add batch to results
      $R_0 \leftarrow R_0 \setminus B$                     ▷ Discard batch from initial ranking
      $F \leftarrow F \setminus B$                            ▷ Discard batch from frontier
      $F \leftarrow F \cup (\text{Neighbours}(B, G) \setminus R_1)$           ▷ Update frontier
      $P \leftarrow \begin{cases} R_0 & \text{if } P = F \\ F & \text{if } P = R_0 \end{cases}$   ▷ Alternate initial ranking and frontier
  while $|R_1| < c$
  $R_1 \leftarrow R_1 \cup \text{Backfill}(R_0, R_1)$              ▷ Backfill remaining items
```

This analysis suggests that the clustering hypothesis holds on TREC DL 2019. Therefore, it follows that the neighbours of passages that a scoring function considers most relevant are a reasonable place to look for additional relevant passages to be scored – which is the core motivation of our proposed method.

## 4 GRAPH-BASED ADAPTIVE RE-RANKING

We now introduce the document re-ranking scenario, and we present a description of our proposed re-ranking algorithm. Let $R_0$ denote an initial ranked pool of $|R_0|$ documents produced by a first-stage ranker, and let $R_1$ denote a subsequent re-ranked pool of $|R_1| = |R_0|$ documents. A certain number of top ranked documents from the $R_1$ pool will subsequently be returned to the user who issued the query. In re-ranking, we assume that the documents from $R_0$ are processed in batches of $b$ documents at maximum (the size of the last batch depends on the re-ranking budget). A scoring function Score() takes as input a batch of documents, e.g., the top scoring $b$ documents in $R_0$ and re-scores them according to a specific re-ranking stage implementation. The re-scored batch is added the final re-ranked pool $R_1$, and then removed from the initial ranked pool $R_0$. Note that by setting $b = 1$, we are re-ranking one document at a time, as in classical learning-to-rank scenarios; in contrast, when $b > 1$, we allow for more efficient re-ranking function implementations leveraging advanced hardware, such as GPUs and TPUs.

Since the time available for re-ranking is often small, and given that it is directly proportional to the number of documents re-ranked, the re-ranking process can be provided with a budget $c$, denoting the maximum number of documents to be re-ranked given the proportional time constraint. If the budget does not allow to re-rank all the document in the initial ranked pool, the Backfill function returns the documents in $R_0$ that have not been re-ranked, i.e., not in $R_1$, that are used to fill up the final re-ranked pool $R_1$ to contain all the documents initially included in $R_0$. For example, if $R_0$ contains 1000 documents and, due to the budget, only 100 documents can be re-scored, the 900 top ranked documents in $R_0$ but not re-ranked in $R_1$ are appended to $R_1$ in the same order as in $R_0$, to obtain a re-ranked list of 1000 documents. The uncoloured lines in Alg. 1 illustrate this re-ranking algorithm, which corresponds to the common re-ranking adopted in a pipelined cascading architecture.

In our adaptive re-ranking algorithm, we leverage a corpus graph $G = (V, E)$. This directed graph encodes the similarity between documents, and can be computed offline, using lexical or semantic similarity function between two documents. Every node in $V$ represents a document in the corpus, and every pair of documents may be connected with an edge in $E$, labelled with the documents' similarity. To address the graph's quadratic space (and time) complexity, we limit to a small value $k$ the number of edges for each node in the corpus graph, i.e., $|E| = k|V|$. The top $k$ edges are selected according to their similarity scores, in decreasing order.

Our adaptive re-ranking algorithm, illustrated in Alg. 1, receives an initial ranking pool of documents $R_0$, a batch size $b$, a budget $c$, and the corpus graph $G$ as input. We consider a dynamically updated re-ranking pool $P$, initialised with the contents of $R_0$ ($P \leftarrow R_0$), and a dynamically updated graph frontier $F$, initially empty ($F \leftarrow \emptyset$). After the re-ranking of the top $b$ documents selected from $P$ and subject to the constraint $c$ (called batch $B$, where $b = |b|$), we update the initial and re-ranked pools $R_0$ and $R_1$. The documents in the batch are removed from the frontier $F$ because there is no need to re-rank them again. Now we consider the documents in the batch $B$, and we look up in the corpus graph for documents whose nodes are directly connected to the documents in $B$. These documents (except any that have already been scored) are added to the frontier ($F \cup (\textsc{Neighbours}(B, G) \setminus R_1)$), prioritised by the computed ranking score of the source document. Note that the neighbours may occur later in the ranking list. Next, instead of using the current contents of the initial pool $R_0$ for the next batch evaluation, we alternate between $R_0$ and the current frontier $F$. In doing so, we ensure that $R_1$ contains documents from $R_0$ and newly identified documents not included in $R_0$. The algorithm proceeds alternating between these two options, populating the frontier at each step, until the budget allows, then backfills the final pool of initial candidates as before.

We note that alternating between the initial ranking and the frontier is somewhat naïve; perhaps it is better to score more/fewer documents from the frontier, or to dynamically decide whether to select batches from the frontier or the initial ranking based on recent scores. Indeed, we investigated such strategies in pilot experiments but were unable to identify a strategy that consistently performed better than the simple alternating technique. We therefore decided to leave the exploration of alternative techniques to future work.

## 5 EXPERIMENTAL SETUP

We experiment to answer the following research questions:

**RQ1** What is the impact of Gar on retrieval effectiveness compared to typical re-ranking?

**RQ2** What is the computational overhead introduced by Gar? (Section 6.2)

**RQ3** How sensitive is Gar to the parameters it introduces: the number of neighbours included in the corpus graph $k$ and the batch size $b$? (Section 6.3)

**RQ4** What is the impact of Gar on retrieval effectiveness compared to state-of-the-art neural IR systems?

Finally, because Gar is based on scoring similar documents, we recognise that it has the potential to reduce the diversity of the retrieved passages (i.e., it could make the retrieved passages more homogeneous). Therefore, we ask:

**RQ5** Does Gar result in more homogeneous relevant passages than existing techniques?

### 5.1 Datasets and Evaluation

Our primary experiments are conducted using the TREC Deep Learning 2019 (DL19) and 2020 (DL20) test collections [3]. DL19 is used throughout the development and for the analysis of Gar, and therefore acts as our validation set. DL20 is held out until the final evaluation, allowing us to confirm that our approach has not over-fit to DL19. Both datasets use the MS MARCO passage ranking corpus, which consists of 8.8M passages [2]. DL19 consists of 43 queries and an average of 215 relevance assessments per query; DL20 has 54 queries with 211 assessments per query. We evaluate our approach using nDCG, MAP, and Recall at rank 1000. For the binary measures (MAP and Recall), we use the standard practice of setting a minimum relevance score of 2, which counts answers that are highly or perfectly relevant. In our experiments we are concerned with both precision and recall, so we focus on nDCG without a rank cutoff, though we also report the official task measure of nDCG with a rank cutoff of 10 (nDCG@10) to provide meaningful comparisons with other works.

We select DL19 and DL20 because they provide more complete relevance assessments than the MS MARCO development set; this is especially important given that Gar is designed to retrieve documents that were not necessarily in the initial re-ranking pool. For completeness, we also report performance on the small subset of MS MARCO dev, which consists of 6980 queries, each with 1.1 relevance assessments per query on average. For this dataset, we report the official measure of Mean Reciprocal Rank at 10 (MRR@10) and the commonly-reported value of Recall at 1000.

### 5.2 Retrieval and Scoring Models

To test the effect of Gar under a variety of initial ranking conditions, we conduct experiments using four retrieval functions as first stage rankers, each representing a different family of ranking approaches.

- **BM25**, a simple and long-standing lexical retrieval approach. We retrieve the top 1000 BM25 results from a PISA [27] index using default parameters.
- **TCT**, a dense retrieval approach. We conduct exact (i.e., exhaustive) retrieval of the top 1000 results using a TCT-ColBERT-HNP model [19] trained on MS MARCO.[3] This is among the most effective dense retrieval models to date.
- **D2Q**, a document expansion approach. We retrieve the top 1000 BM25 results from a PISA index of documents expanded using a docT5query model [30] trained on MS MARCO. We use the expanded documents released by the authors. This is the most effective document expansion model we are aware of to date.
- **SPLADE**, a learned sparse lexical retrieval model. We retrieve the top 1000 results for a SPLADE++ model [7] trained on MS MARCO (`CoCondenser-EnsembleDistil` version). We use code released by the authors for indexing and retrieval.[4] This is the most effective learned lexical retrieval model we are aware of to date.

Similarly, we experiment with the following neural re-ranking models to test the effect of the scoring function on Gar.

- **MonoT5**, a sequence-to-sequence scoring function. We test two versions of the MonoT5 model [29] trained on MS MARCO from two base language models: MonoT5-base, and MonoT5-3b. The 3b model has the same structure as the base model, but has more

[3]Hugging Face ID: `castorini/tct_colbert-v2-hnp-msmarco`
[4]https://github.com/naver/splade

parameters (13× more; 2.9B, compared to base's 223M) so it is consequently more expensive to run. These models are among the most effective scoring functions reported to date.[5]

- **ColBERT** (scorer only), a late interaction scoring function. Although ColBERT [12] can be used in an end-to-end fashion (i.e., using its embeddings to perform dense retrieval), we use it as a scoring function over the aforementioned retrieval functions. The model represents two paradigms: one where representations are pre-computed to reduce the query latency, and another where the representations are computed on-the-fly.

We use the implementations of the above methods provided by PyTerrier [24]. Following PyTerrier notation, we use » to denote a re-ranking pipeline. For instance, "BM25»MonoT5-base" retrieves using BM25 and re-ranks using MonoT5-base.

## 5.3 Corpus Graphs

In our experiments, we construct and exploit two corpus graphs, namely a lexical similarity graph and a semantic similarity graph. The lexical graph (denoted as $\text{GAR}_{\text{BM25}}$) is constructed by retrieving the top BM25 [34] results using the text of the passage as the query. We use PISA to perform top $k + 1$ lexical retrieval (discarding the passage itself). Using a 4.0 GHz 24-core AMD Ryzen Threadripper Processor, the MS MARCO passage graph takes around 8 hours to construct. The semantic similarity graph (denoted as $\text{GAR}_{\text{TCT}}$) is constructed using the TCT-ColBERT-HNP model. We perform an exact (i.e., exhaustive) search over an index to retrieve the top $k + 1$ most similar embeddings to each passage (discarding the passage itself). Using an NVIDIA GeForce RTX 3090 GPU to compute similarities, the MS MARCO passage graph takes around 3 hours to construct.

We construct both graphs using $k = 8$ neighbours, and explore the robustness to various values of $k$ in Section 6.3. Because the number of edges (i.e., neighbours) per node (i.e., passage) is known, the graphs are both stored as a uncompressed sequence of docids. Using unsigned 32-bit integer docids, only 32 bytes per passage are needed, which amounts to 283 MB to store an MS MARCO graph.[6] We note that there are likely approaches that reduce the computational overhead in graph construction by making use of approximate searches; we leave this for future work. The two graphs differ substantially in their content.[7] We release these graphs through our implementation to aid other researchers and enable future works.

## 5.4 Other Parameters and Settings

We use a GAR batch size of $b = 16$ by default, matching a typical batch size for a neural cross-encoder model. We explore the robustness of GAR to various values of $b$ in Section 6.3. We explore two budgets: $c = 100$ (a reasonable budget for a deployed re-ranking system, e.g., [9]) and $c = 1000$ (the *de facto* default threshold commonly used in shared tasks like TREC).

## 6 RESULTS AND ANALYSIS

We now present the results of our experiments and conduct associated analysis to answer our research questions.

## 6.1 Effectiveness

To understand whether GAR is generally effective, it is necessary to test the effect it has on a variety of retrieval pipelines. Therefore, we construct re-ranking pipelines based on every pair of our initial ranking functions (BM25, TCT, D2Q, and SPLADE) and scoring functions (MonoT5-base, MonoT5-3b, and ColBERT). These 12 pipelines collectively cover a variety of paradigms. Table 2 presents the results of GAR on these pipelines for TREC DL 2019 and 2020 using both the lexical BM25-based graph and the semantic TCT-based corpus graph. We report results using both re-ranking budgets $c = 100$ and $c = 1000$.

Each box in Table 2 allows the reader to inspect the effect on retrieval effectiveness that GAR has on a particular re-ranking pipeline and re-ranking budget. In general, we see that the greatest improvement when the initial retrieval pool is poor. In particular, BM25 only provides a R@1k of 0.755 and 0.805 on DL19 and DL20, respectively, while improved retrieval functions offer up to 0.872 and 0.899, respectively (SPLADE). GAR enables the pipelines to find additional relevant documents. Using BM25 as the initial pool, our approach reaches a R@1k up to 0.846 and 0.892, respectively (BM25»MonoT5-3b w/ $\text{GAR}_{\text{TCT}}$ and $c = 1000$). Perhaps unsurprisingly, this result is achieved using both a corpus graph ($\text{GAR}_{\text{TCT}}$) that differs substantially from the technique used for initial retrieval (BM25) and using the most effective re-ranking function (MonoT5-3b). However, we also note surprisingly high recall in this setting when using the $\text{GAR}_{\text{BM25}}$ corpus graph: up to 0.831 (DL19) and 0.881 (DL20). These results are on par with the recall achieved by TCT and D2Q – an impressive feat considering that this pipeline only uses lexical signals and a single neural model trained with a conventional process.[8] The pipelines that use a BM25 initial ranker also benefit greatly in terms of nDCG, which is likely due in part to the improved recall.

Significant improvements are also observed in all other pipelines, particularly in terms of nDCG when there is a low re-ranking budget available ($c = 100$) and in recall when a high budget is available ($c = 1000$). In general, the corpus graph that is least similar to the initial ranker is most effective (e.g., the BM25 graph when using a TCT ranking). However, we note that both corpus graphs improve every pipeline, at least in some settings. For instance, the $\text{GAR}_{\text{TCT}}$ corpus graph consistently improves the nDCG of pipelines that use TCT as an initial ranker, but rarely the recall.

We also note that GAR can nearly always improve the precision of the top results, as measured by nDCG, in settings with a limited re-ranking budget ($c = 100$), even when R@1k remains unchanged. This is likely due to the fact that GAR is able to pick out documents from lower depths of the initial ranking pool to score within the limited available budget. For instance, in the case of the strong SPLADE»MonoT5-base pipeline with $c = 100$, which offers high recall to begin with (0.872 on DL19 and 0.899 on DL20), $\text{GAR}_{\text{BM25}}$ improves the nDCG from 0.750 to 0.762 (DL19) and from 0.748 to 0.757 (DL20), while leaving the R@1k unchanged.

In a few rare cases, we observe that GAR can yield a lower mean performance than the baseline (e.g., MAP for the D2Q»MonoT5-base pipeline with $c = 1000$). However, these differences are never

[5]We also experiment with applying DuotT5 [32] as a final re-ranker in Section 6.4.
[6]For context, the compressed document source is 1035MB, and the compressed PISA index of MS MARCO is 647MB.
[7]Only 3% of passages agree on seven or eight neighbours across graphs, and 43% of passages have no agreement on neighbours across graphs.

[8]D2Q is trained as a sequence-to-sequence model and involves a lengthy inference stage during indexing, while TCT employs a complex, multi-stage training process involving another trained scoring model that is challenging to fully reproduce [38]. Meanwhile, MonoT5-3b is simply trained using MS MARCO's training triples.

**Table 2: Effectiveness of Gar on TREC DL 2019 and 2020 in a variety of re-ranking pipelines and re-ranking budgets ($c$). The top result for each pipeline is in bold. Significant differences with the baseline (typical re-ranking) are marked with *, while insignificant differences are in grey (paired t-test, $p < 0.05$, using Bonferroni correction).**

| | DL19 (valid.) $c = 100$ | | | DL19 (valid.) $c = 1000$ | | | DL20 (test) $c = 100$ | | | DL20 (test) $c = 1000$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pipeline | nDCG | MAP | R@1k | nDCG | MAP | R@1k | nDCG | MAP | R@1k | nDCG | MAP | R@1k |
| BM25»MonoT5-base | 0.665 | 0.417 | 0.755 | 0.699 | 0.483 | 0.755 | 0.672 | 0.421 | 0.805 | 0.711 | 0.498 | 0.805 |
| w/ Gar$_{\text{BM25}}$ | * 0.697 | * 0.456 | * 0.786 | 0.727 | 0.490 | * 0.827 | * 0.695 | 0.439 | * 0.823 | * 0.743 | **0.501** | * 0.874 |
| w/ Gar$_{\text{TCT}}$ | * **0.722** | * **0.491** | * **0.800** | * **0.743** | **0.511** | * **0.839** | * **0.714** | * **0.472** | * **0.831** | * **0.749** | **0.501** | * **0.892** |
| BM25»MonoT5-3b | 0.667 | 0.418 | 0.755 | 0.700 | 0.489 | 0.755 | 0.678 | 0.442 | 0.805 | 0.728 | 0.534 | 0.805 |
| w/ Gar$_{\text{BM25}}$ | * 0.693 | 0.454 | * 0.790 | * 0.741 | 0.517 | * 0.831 | * 0.715 | * 0.469 | * 0.829 | * 0.772 | 0.556 | * 0.881 |
| w/ Gar$_{\text{TCT}}$ | * **0.715** | * **0.484** | * **0.806** | * **0.746** | **0.522** | * **0.846** | * **0.735** | * **0.512** | * **0.837** | * **0.787** | * **0.564** | * **0.899** |
| BM25»ColBERT | 0.663 | 0.409 | 0.755 | 0.681 | 0.458 | 0.755 | 0.667 | 0.421 | 0.805 | 0.697 | 0.469 | 0.805 |
| w/ Gar$_{\text{BM25}}$ | * 0.690 | * 0.442 | * 0.783 | * 0.720 | 0.480 | * 0.825 | * 0.695 | * 0.446 | * 0.823 | * 0.732 | 0.479 | * 0.870 |
| w/ Gar$_{\text{TCT}}$ | * **0.716** | * **0.475** | * **0.798** | * **0.727** | **0.482** | * **0.841** | * **0.707** | * **0.463** | * **0.829** | * **0.740** | **0.481** | * **0.887** |
| TCT»MonoT5-base | 0.708 | 0.472 | 0.830 | 0.704 | 0.473 | 0.830 | 0.698 | 0.488 | 0.848 | 0.693 | 0.471 | 0.848 |
| w/ Gar$_{\text{BM25}}$ | * **0.728** | **0.484** | **0.852** | * **0.733** | **0.480** | * **0.883** | * **0.719** | * **0.501** | **0.861** | * **0.719** | **0.473** | * **0.881** |
| w/ Gar$_{\text{TCT}}$ | 0.722 | 0.481 | 0.847 | * 0.724 | 0.474 | 0.866 | * 0.712 | 0.494 | 0.856 | * 0.710 | 0.471 | 0.871 |
| TCT»MonoT5-3b | 0.720 | 0.498 | 0.830 | 0.725 | 0.513 | 0.830 | 0.723 | 0.534 | 0.848 | 0.733 | 0.544 | 0.848 |
| w/ Gar$_{\text{BM25}}$ | * **0.748** | * **0.521** | * **0.857** | * **0.759** | **0.521** | * **0.885** | * **0.743** | **0.546** | * **0.864** | * **0.771** | * **0.555** | * **0.890** |
| w/ Gar$_{\text{TCT}}$ | * 0.742 | * 0.517 | 0.849 | * 0.749 | 0.516 | * 0.868 | * 0.741 | * 0.545 | * 0.861 | * 0.759 | 0.551 | * 0.880 |
| TCT»ColBERT | 0.708 | 0.464 | 0.830 | 0.701 | 0.452 | 0.830 | 0.698 | 0.476 | 0.848 | 0.697 | 0.470 | 0.848 |
| w/ Gar$_{\text{BM25}}$ | * **0.729** | * **0.480** | **0.853** | * **0.727** | **0.459** | **0.876** | * **0.715** | **0.485** | **0.857** | * **0.722** | * **0.477** | * **0.877** |
| w/ Gar$_{\text{TCT}}$ | * 0.722 | 0.474 | 0.845 | * 0.715 | 0.452 | 0.852 | * 0.711 | * 0.484 | * **0.857** | * 0.713 | 0.473 | 0.864 |
| D2Q»MonoT5-base | 0.736 | 0.503 | 0.830 | 0.747 | 0.531 | 0.830 | 0.726 | 0.499 | 0.839 | 0.731 | **0.508** | 0.839 |
| w/ Gar$_{\text{BM25}}$ | * 0.748 | 0.506 | 0.848 | 0.757 | 0.519 | * **0.880** | * 0.734 | 0.497 | * 0.847 | **0.748** | 0.504 | * 0.880 |
| w/ Gar$_{\text{TCT}}$ | * **0.760** | * **0.528** | **0.850** | * **0.766** | **0.533** | * 0.879 | **0.740** | **0.508** | * **0.856** | **0.748** | 0.499 | * **0.895** |
| D2Q»MonoT5-3b | 0.737 | 0.506 | 0.830 | 0.751 | 0.542 | 0.830 | 0.738 | 0.531 | 0.839 | 0.753 | 0.557 | 0.839 |
| w/ Gar$_{\text{BM25}}$ | 0.744 | 0.512 | * 0.850 | **0.772** | **0.549** | * **0.880** | * 0.751 | 0.535 | * 0.852 | * 0.781 | 0.561 | * 0.887 |
| w/ Gar$_{\text{TCT}}$ | **0.755** | **0.524** | * **0.857** | 0.769 | 0.544 | * **0.880** | * **0.764** | **0.550** | * **0.860** | * **0.790** | **0.565** | * **0.905** |
| D2Q»ColBERT | 0.724 | 0.475 | 0.830 | 0.733 | 0.501 | 0.830 | 0.718 | 0.483 | 0.839 | 0.717 | 0.479 | 0.839 |
| w/ Gar$_{\text{BM25}}$ | 0.734 | 0.484 | 0.845 | **0.753** | **0.505** | * 0.876 | * 0.731 | 0.487 | * 0.849 | * 0.737 | 0.482 | * 0.872 |
| w/ Gar$_{\text{TCT}}$ | * **0.744** | * **0.496** | **0.849** | * 0.752 | 0.503 | * **0.878** | * **0.735** | **0.488** | * **0.856** | * **0.746** | **0.485** | * **0.893** |
| SPLADE»MonoT5-base | 0.750 | 0.506 | 0.872 | 0.737 | **0.487** | 0.872 | 0.748 | 0.505 | 0.899 | 0.731 | **0.480** | 0.899 |
| w/ Gar$_{\text{BM25}}$ | * **0.762** | 0.509 | **0.888** | **0.745** | **0.487** | **0.893** | * **0.757** | **0.509** | 0.902 | **0.737** | 0.479 | **0.909** |
| w/ Gar$_{\text{TCT}}$ | * 0.759 | **0.512** | 0.878 | 0.737 | 0.481 | 0.875 | 0.751 | 0.506 | **0.903** | 0.734 | 0.475 | 0.908 |
| SPLADE»MonoT5-3b | 0.761 | 0.526 | 0.872 | 0.764 | **0.533** | 0.872 | 0.774 | 0.559 | 0.899 | 0.775 | 0.560 | 0.899 |
| w/ Gar$_{\text{BM25}}$ | * **0.775** | 0.532 | * **0.891** | **0.774** | **0.533** | **0.896** | * **0.780** | 0.559 | 0.903 | * **0.788** | **0.562** | * **0.919** |
| w/ Gar$_{\text{TCT}}$ | * 0.773 | **0.539** | 0.884 | 0.769 | 0.531 | 0.881 | * **0.780** | **0.561** | **0.905** | 0.783 | 0.559 | 0.910 |
| SPLADE»ColBERT | 0.741 | 0.479 | 0.872 | 0.727 | **0.456** | 0.872 | 0.747 | 0.495 | 0.899 | 0.733 | 0.474 | 0.899 |
| w/ Gar$_{\text{BM25}}$ | * **0.753** | **0.490** | **0.885** | **0.730** | **0.456** | **0.875** | * **0.755** | **0.501** | 0.902 | * **0.742** | * **0.477** | **0.914** |
| w/ Gar$_{\text{TCT}}$ | * 0.750 | 0.489 | 0.876 | 0.727 | 0.455 | 0.868 | * 0.752 | 0.500 | **0.903** | 0.740 | * 0.476 | 0.911 |

statistically significant and are usually accompanied by significant improvements to other measures (e.g., the R@1k improves).

We note that the same trends appear for both our validation set (DL19) and our held-out test set (DL20), suggesting that Gar is not over-fitted to the data that we used during the development of Gar.

Finally, we test Gar on the MS MARCO dev (small) set. This setting differs from the TREC DL experiments in that each of the queries has only a few (usually just one) passages that are labeled as relevant, but has far more queries (6,980 compared to 43 in DL19 and 54 in DL20). Thus, experiments on this dataset test a pipeline's capacity to retrieve a single (and somewhat arbitrary) relevant passage for a query.[9] Due to the cost of running multiple versions of highly-expensive re-ranking pipelines, we limit this study to a low re-ranking budget $c = 100$ and to the two less expensive scoring functions (MonoT5-base and ColBERT). Table 3 presents the results. We find that Gar offers the most benefit in pipelines that suffer from the lower recall – namely, the BM25-based pipelines. In this setting, the improved R@1k also boosts the RR@10. In the TCT, D2Q, and SPLADE pipelines, R@1k often significantly improved, but this results in non-significant (or marginal) changes to RR@10.

To answer RQ1, we find that Gar provides significant benefits in terms of precision- and recall-oriented measures. The results hold across a variety of initial retrieval functions, re-ranking functions, and re-ranking budgets. The most benefit is apparent when the initial pool has low recall, though we note that Gar also improves over systems with high initial recall – particularly by enabling

[9]The suitability of this dataset for evaluation is debated in the community (e.g., [1, 25]), but we include it for completeness.

**Table 3: Effectiveness of Gar on the MS MARCO dev (small) set with a re-ranking budget of $c = 100$. The top result for each pipeline is in bold. Significant differences with the baseline (typical re-ranking) are marked with * (paired t-test, $p < 0.05$, using Bonferroni correction).**

| | »MonoT5-base | | »ColBERT | |
|---|---|---|---|---|
| Pipeline | RR@10 | R@1k | RR@10 | R@1k |
| BM25» | 0.356 | 0.868 | 0.323 | 0.868 |
| $\text{Gar}_{\text{BM25}}$ | 0.358 | * 0.881 | 0.323 | * 0.882 |
| $\text{Gar}_{\text{TCT}}$ | ***0.369** | ***0.903** | ***0.333** | ***0.902** |
| TCT» | 0.388 | 0.970 | 0.345 | 0.970 |
| $\text{Gar}_{\text{BM25}}$ | **0.389** | ***0.973** | ***0.346** | ***0.973** |
| $\text{Gar}_{\text{TCT}}$ | 0.388 | ***0.973** | **0.346** | * 0.972 |
| D2Q» | **0.386** | 0.936 | **0.345** | 0.936 |
| $\text{Gar}_{\text{BM25}}$ | **0.386** | * 0.941 | **0.345** | * 0.941 |
| $\text{Gar}_{\text{TCT}}$ | **0.386** | ***0.949** | 0.344 | ***0.948** |
| SPLADE» | **0.389** | 0.983 | 0.345 | 0.983 |
| $\text{Gar}_{\text{BM25}}$ | **0.389** | **0.984** | ***0.346** | **0.984** |
| $\text{Gar}_{\text{TCT}}$ | 0.388 | ***0.984** | ***0.346** | **0.984** |

higher precision at a lower re-ranking budget. Overall, we find that Gar is safe to apply to any re-ranking pipeline (i.e., it will not harm the effectiveness), and it will often improve performance (particularly when the re-ranking budget is limited or when a low-cost first stage retriever is used).

To illustrate the ability of Gar to promote low-ranked documents under limited ranking budgets, Figure 2 plots the initial rank (x-axis) of documents and their final rank (y-axis), for a particular query. Each point represents a retrieved document, with colour/size indicative of the relevance label. Lines between points indicate links followed in the corpus graph. It can be seen that by leveraging the corpus graph, Gar is able to promote highly relevant documents that were lowly scored in the initial ranking, as well as retrieve 'new' relevant documents, which are not retrieved in the initial BM25 pool. For instance, GAR is able to select five rel=2 documents from around initial rank 250-300, and ultimately score them within the top 40 documents. Meanwhile, it retrieves two rel=2 and one rel=3 documents that were not found in the first stage.

## 6.2 Computational Overhead

Gar is designed to have a minimal impact on query latency. By relying on a pre-computed corpus graph that will often be small enough to fit into memory (283MB with $k = 8$ for MS MARCO), neighbour lookups are performed in $O(1)$ time. With the frontier $F$ stored in a heap, insertions take only $O(1)$, meaning that finding neighbours and updating the frontier adds only a constant time for each scored document. Sampling the top $b$ items from the heap takes $O(b \log c)$, since the number of items in the heap never needs to exceed the budget $c$.

To obtain a practical sense of the computational overhead of Gar, we conduct latency tests. To isolate the effect of Gar itself, we find it necessary to factor out the overhead from the re-ranking model itself, since the variance in latency between neural scoring runs often exceeds the overhead introduced by Gar. To this end, we

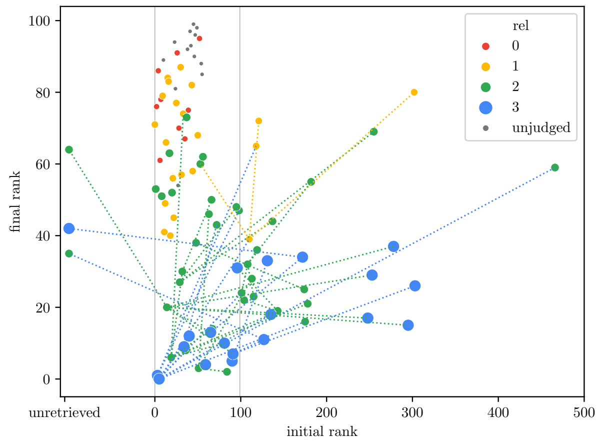

**Figure 2: Plot of the initial and final rankings of BM25»MonoT5-base using $\text{Gar}_{\text{TCT}}$ with $c = 100$ for the DL19 query *'how long is life cycle of flea'*. The colour/size of dots indicate the relevance label. Lines between points indicate links followed in the corpus graph.**

pre-compute and store all the needed query-document scores and simply look them up as they would be scored. We then test various re-ranking budgets ($c$) for DL19, and take 10 latency measurements of the typical re-ranking and Gar processes. Table 4 reports the differences between the latency of Gar and the typical re-ranking results, isolating the overhead of Gar itself. We find that Gar introduces less than 37.37ms overhead per 1000 documents scores (i.e., 2.68-3.73ms overhead per 100 documents scored), on average, using 16 documents per batch. We report results using the semantic TCT-based corpus graph, though we find little difference when using the lexical BM25-based corpus graph. The overhead can be further reduced (down to 3.1ms per 100 documents) by using a larger batch size, i.e., 64 documents per batch; we explore the effect of the batch size parameter on effectiveness in Section 6.3. When compared to the cost of monoT5 scoring (rightmost column in Table 4), the Gar process adds negligible overhead, typically amounting to less than a 2% increase in latency and falls within the variance of the scoring function's latency for low re-ranking budgets.

This experiment answers RQ2: the online computational overhead of Gar is minimal. It can be efficiently implemented using a heap, and adds only around 3-4ms per 100 documents in the re-ranking budget. This overhead is negligible when compared with the latency of a leading neural scoring function, though it will represent a higher proportion for more efficient scoring functions.

## 6.3 Robustness to Parameters

Recall that Gar introduces two new parameters: the number of nearest neighbours in the corpus graph $k$ and the batch size $b$. In this section, we conduct experiments to test whether Gar is robust to the settings of these parameters.[10] We separately sweep $k \in [1, 16]$ and $b \in [1, 512]$ (by powers of 2) over DL19 with $c = 1000$ for all Gar pipelines, and present the different effectiveness metrics in Figure 3.

[10]Due to the number of pipelines and parameter settings, an exhaustive grid search over these parameters is prohibitively expensive.

With regard to the number of graph neighbours $k$, the nDCG, MAP and recall metrics are relatively stable from around $k = 6$ to $k = 16$ for almost all pipelines. The MAP performance appears to be the least stable in this range, with some fluctuations in performance between $k = 7$ and $k = 13$. Recall appears to be most affected, with sharp gains for some pipelines between $k = 1$ to $k = 4$. This trend is present also for nDCG.

The batch size $b$ is remarkably stable from $b = 1$ to $b = 128$, with only a blip in effectiveness for the BM25 graph at $b = 16$. The most prominent shift in performance occurs at large batch sizes, e.g., $b = 512$. We note that, when $b = 512$, the corpus graph can only be traversed for a single hop – the neighbours of the top-scoring documents from the frontier batch are not able to be fed back into the re-ranking pool. This validates our technique of incorporating the feedback mechanism into the re-ranking process itself, which gives the model more chances to traverse the graph. While it may be tempting to prefer the stability of the system with very low batch sizes, we note that this has an effect on the performance: as seen in Section 6.2, lower batch sizes reduces the speed of Gar itself. Further, and more importantly, $b$ imposes a maximum batch size of the scoring function itself; given that neural models benefit considerably in terms of performance with larger batch sizes (since the operations on the GPU are parallelised), larger values of $b$ (e.g., $b = 16$ to $b = 128$) should be preferred for practical reasons.

To answer RQ3, we find that the performance of Gar is stable across various pipelines when the number of neighbours is sufficiently large ($k \geq 6$) and the batch size is sufficiently low ($b \leq 128$).

## 6.4 Baseline Performance

Section 6.1 established the effectiveness of Gar as ablations over a variety of re-ranking pipelines. We now explore how the approach fits into the broader context of the approaches proposed for passage retrieval and ranking. We explore two classes of pipelines: 'Kitchen Sink' approaches that combine numerous approaches and models together, and 'Single-Model' approaches that use only involve a single neural model at any stage. We select representative Gar variants based on the nDCG@10 performance on DL19 (i.e., as a validation set), with DL20 again treated as the held-out test set. All systems use a re-ranking budget of $c = 1000$. In this table, we report nDCG@10 to allow comparisons against prior work. We also report the judgment rate at 10 to provide context about how missing information in the judgments may affect the nDCG@10 scores.

The Kitchen Sink results are reported in the top section of Table 5. All systems involve three ranking components: an initial retriever $R_0$, a Mono-scorer $R_1$ (which assigns a relevance score to each document), and a Duo-scorer $R_2$ (which scores and aggregates pairs of documents). The Duo-style models are known to improve the ranking of the top documents [32]. Although we leave the exploration of how Gar can be used to augment the Duo process directly for future work, we still want to check what effect Gar has on these pipelines. We ablate two Duo systems (either based on D2Q or SPLADE) using Gar for the first-stage re-ranker and a DuoT5-3b-based second-stage re-ranker (second stage uses the suggested cutoff of 50 from [32]). We observe that there is no significant difference in terms of precision of the top 10 results. However, Gar can still provide a significant improvement in terms of nDCG later in the ranking and in terms of recall. These results suggest that although Gar identifies more relevant documents, the Duo models are not capable of promoting them to the top ranks.

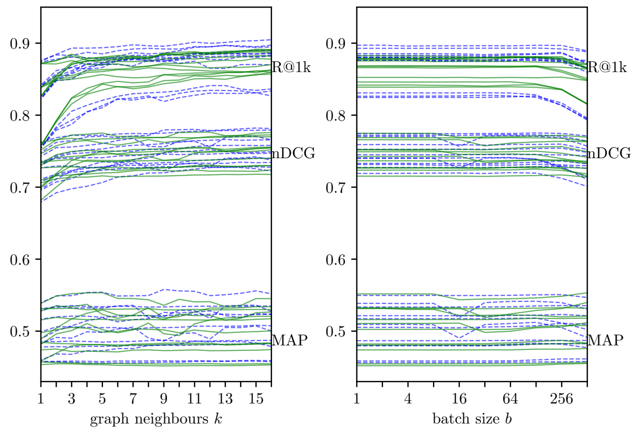

**Figure 3: Performance of Gar when the number of neighbours in the corpus graph $k$ and the batch size $b$ vary. Each line represents a system from Table 2. The dashed blue (solid green) lines are for the BM25 (TCT) graph.**

**Table 4: Mean latency overheads (ms/query) for Gar with 95% confidence intervals. The latency of MonoT5-base scoring (with a model batch size of 64) is presented for context.**

| | $\text{Gar}_{\text{TCT}}$ | | MonoT5-base |
|---|---|---|---|
| c | $b = 16$ | $b = 64$ | Scoring |
| 100 | $2.68 \pm 0.02$ | $0.57 \pm 0.01$ | $267.06 \pm 6.12$ |
| 250 | $8.10 \pm 0.05$ | $4.34 \pm 0.01$ | $652.30 \pm 7.53$ |
| 500 | $17.38 \pm 0.07$ | $13.66 \pm 0.02$ | $1,362.14 \pm 5.27$ |
| 750 | $26.96 \pm 0.12$ | $22.29 \pm 0.07$ | $2,047.20 \pm 6.71$ |
| 1000 | $37.37 \pm 0.07$ | $30.82 \pm 0.04$ | $2,631.75 \pm 6.28$ |

We next explore Single-Model systems, which are shown in the bottom section of Table 5. Having only a single models likely has some practical advantages: pipelines that use a single model tend to be simpler, and practitioners only need to train a single model. Here, we compare with a variety of systems that fall into this category, most notably the recently-proposed ColBERT-PRF approaches that operate over dense indexes [39]. A $\text{Gar}_{\text{BM25}}$ pipeline that operates over BM25 results also falls into this category, since only a single neural model (the scorer) is needed. Among this group, Gar performs competitively, outmatched only by ColBERT-PRF [39] and the recent SPLADE [7] model (though the differences in performance are not statistically significant). Compared to these methods, though, Gar requires far less storage – the corpus graph for Gar is only around 283MB, while the index for SPLADE is 8GB, and the vectors required for ColBERT-PRF are 160GB.

To answer RQ4: We observe that Gar can be incorporated into a variety of larger, state-of-the-art re-ranking pipelines. It frequently boosts the recall of systems that it is applied to, though the scoring functions we explore tend to have difficulty in making use of the additional relevant passages. This motivates exploring further improvements to re-ranking models. For instance, cross-encoder

**Table 5: Performance of GAR compared to a variety of other baselines. Significant differences are computed within groups, with significance denoted as superscript letters $^{a-c}$ (paired t-test, $p < 0.05$, Bonferroni correction). Rows marked with † are given to provide additional context, but the metrics were copied from other papers so do not include statistical tests.**

| | | | | | DL19 (validation) | | | | DL20 (test) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $R_0$ | $R_1$ | $R_2$ | RR | nDCG@10 | nDCG | R@1k | Judged@10 | nDCG@10 | nDCG | R@1k | Judged@10 |
| **Kitchen Sink Systems** | | | | | | | | | | | | |
| | D2Q | »MonoT5-3b | »DuoT5-3b | | **0.771** | 0.756 | $^{ab}$0.830 | 0.958 | 0.785 | $^{ab}$0.754 | $^{ab}$0.839 | 0.996 |
| | SPLADE | »MonoT5-3b | »DuoT5-3b | | 0.768 | 0.772 | 0.872 | 0.953 | 0.787 | $^{a}$0.781 | $^{a}$0.899 | 0.987 |
| a | SPLADE | »MonoT5-3b | »DuoT5-3b | GAR$_{BM25}$ | 0.767 | **0.781** | **0.896** | 0.951 | 0.787 | **0.794** | **0.919** | 0.989 |
| b | D2Q | »MonoT5-3b | »DuoT5-3b | GAR$_{TCT}$ | 0.766 | 0.775 | 0.880 | 0.953 | **0.788** | 0.793 | 0.905 | 0.993 |
| † | TAS-B+D2Q [8] | »MonoT5-3b | »DuoT5-3b | | 0.759 | - | 0.882 | - | 0.783 | - | 0.895 | - |
| **Single-Model Systems** | | | | | | | | | | | | |
| | ColBERT ANN | »ColBERT | »ColBERT-PRF | | **0.739** | **0.764** | 0.871 | 0.907 | 0.715 | 0.746 | 0.884 | 0.946 |
| | SPLADE | - | - | | 0.731 | 0.755 | **0.872** | 0.926 | 0.720 | 0.750 | **0.899** | 0.970 |
| c | BM25 | »MonoT5-3b | - | GAR$_{BM25}$ | 0.729 | 0.741 | 0.831 | 0.947 | **0.756** | **0.772** | 0.881 | 0.972 |
| | BM25 | »MonoT5-3b | - | | 0.722 | 0.700 | $^{c}$0.755 | 0.944 | 0.749 | $^{c}$0.728 | $^{c}$0.805 | 0.980 |
| | TCT | - | - | | 0.721 | 0.708 | 0.830 | 0.914 | $^{c}$0.686 | $^{c}$0.689 | 0.848 | 0.931 |
| | ColBERT ANN | »ColBERT | - | | 0.693 | 0.687 | 0.789 | 0.884 | $^{c}$0.687 | $^{c}$0.711 | 0.825 | 0.937 |
| † | ANCE | - | - | | 0.648 | - | 0.755 | 0.851 | 0.646 | - | 0.776 | 0.865 |
| | D2Q | - | - | | $^{c}$0.615 | $^{c}$0.678 | 0.830 | 0.916 | $^{c}$0.608 | $^{c}$0.676 | 0.839 | 0.956 |

**Table 6: Intra-List Similarity (ILS) among retrieved relevant documents. Since the set of retrieved documents does not change using typical Re-Ranking (RR), each value in this column is only listed once. ILS scores that are statistically equivalent to the RR setting are indicated with * (procedure described in Section 6.5).**

| | | GAR$_{BM25}$ | | GAR$_{TCT}$ | |
|---|---|---|---|---|---|
| Pipeline | RR | $c$=100 | $c$=1k | $c$=100 | $c$=1k |
| BM25»MonoT5-base | 0.947 | * 0.946 | * 0.946 | * 0.947 | * 0.946 |
| BM25»MonoT5-3b | | * 0.946 | * 0.946 | * 0.946 | * 0.946 |
| BM25»ColBERT | | * 0.946 | * 0.946 | * 0.947 | * 0.946 |
| TCT»MonoT5-base | 0.969 | * 0.969 | * 0.968 | * 0.969 | * 0.969 |
| TCT»MonoT5-3b | | * 0.969 | * 0.968 | * 0.969 | * 0.969 |
| TCT»ColBERT | | * 0.969 | * 0.969 | * 0.969 | * 0.969 |
| D2Q»MonoT5-base | 0.969 | * 0.968 | * 0.968 | * 0.969 | * 0.968 |
| D2Q»MonoT5-3b | | * 0.968 | * 0.968 | * 0.968 | * 0.968 |
| D2Q»ColBERT | | * 0.968 | * 0.968 | * 0.969 | * 0.968 |
| SPLADE»MonoT5-base | 0.969 | * 0.968 | * 0.968 | * 0.969 | * 0.969 |
| SPLADE»MonoT5-3b | | * 0.968 | * 0.968 | * 0.968 | * 0.969 |
| SPLADE»ColBERT | | * 0.968 | * 0.969 | * 0.969 | * 0.969 |

models have largely relied on simple BM25 negative sampling (e.g., from the MS MARCO triples file) for training. Techniques like hard negative sampling [40] and distillation [19] (employed to train models like SPLADE and TCT) have so far been largely unexplored for cross-encoder models; these techniques may help them recognise more relevant documents.

## 6.5 Diversity of Retrieved Passages

Next, we test whether GAR results in a more homogeneous set of retrieved relevant passages, compared to typical re-ranking. Among the set of relevant passages each system retrieved,[11] we compute the Intra-List Similarity (ILS) [41] using our TCT embeddings. ILS is the average cosine similarity between all pairs of items in a set, so a higher ILS values here indicate that the relevant documents are more similar to one another. Table 6 compares the ILS of each initial ranking function (BM25, TCT, D2Q, and SPLADE) with the GAR$_{BM25}$ and GAR$_{TCT}$ counterparts. Using two-one-sided t-tests (TOSTs) with bounds of 0.005 and $p < 0.05$ (including a Bonferonni correction), we find that GAR yields statistically equivalent diversity to the typical re-ranking system.

These results answer RQ5: despite using document similarity to help choose additional documents to score, GAR does not result in the system retrieving a more homogeneous set of relevant passages.

[11]We are only concerned with the diversity among the relevant passages ($rel = 2$ or $rel = 3$) because non-relevant passages are inherently dissimilar from relevant ones.

## 7 CONCLUSIONS AND OUTLOOK

In this paper we took a modern approach to the cluster hypothesis, to consider nearest neighbour documents while re-ranking using a neural re-ranker. In this way, the neural re-ranker feedbacks how useful documents are, which allows to prioritise further the next batch of documents to identify in an adaptive manner. Experiments using nearest neighbour graphs computed using BM25 and TCT-ColBERT-HNP demonstrated the promise of our Graph-based Adaptive Re-ranking approach, with significant improvements in precision- and recall-oriented measures on the TREC DL 2019 & 2020 passage ranking corpus. For instance, GAR can improve nDCG by up to 8% (BM25»monoT5-base w/ TCT) and R@1000 up to 12% (also BM25»monoT5-base w/ TCT).

We believe this work opens up several directions for the modelling of adaptive re-ranking - indeed, it can be seen as a *closed loop* modelling problem (as opposed to the classical *open loop* re-ranking formulation), or as an explore/exploit scenario. Due to the effectiveness and efficiency of the instantiations shown here, we leave further advanced formulations to future work.

## ACKNOWLEDGEMENTS

Sean and Craig acknowledge EPSRC grant EP/R018634/1: Closed-Loop Data Science for Complex, Computationally- & Data-Intensive Analytics.

## REFERENCES

[1] Negar Arabzadeh, Alexandra Vtyurina, Xinyi Yan, and Charles L. A. Clarke. 2021. Shallow pooling for sparse labels. *arXiv preprint 2109:00062* (2021).

[2] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *Proc. InCoCo@NIPS*.

[3] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, Ellen M. Voorhees, and Ian Soboroff. 2021. TREC Deep Learning Track: Reusable Test Collections in the Large Data Regime. In *Proc. SIGIR*.

[4] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. 1992. Scatter/Gather: A Cluster-Based Approach to Browsing Large Document Collections. In *Proc. SIGIR*.

[5] Zhuyun Dai and Jamie Callan. 2019. Context-aware sentence/passage term importance estimation for first stage retrieval. *Preprint: arXiv:1910.10687* (2019).

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL*.

[7] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective. *arXiv preprint 2205.04733* (2022).

[8] Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. 2021. Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling. In *Proc. SIGIR*.

[9] Sebastian Hofstätter, Bhaskar Mitra, Hamed Zamani, Nick Craswell, and Allan Hanbury. 2021. Intra-Document Cascading: Learning to Select Passages for Neural Document Ranking. In *Proc. SIGIR*.

[10] Sebastian Hofstätter and Allan Hanbury. 2019. Let's measure run time! Extending the IR replicability infrastructure to include performance aspects. *arXiv preprint 1907:04614* (2019).

[11] Nick Jardine and Cornelis Joost van Rijsbergen. 1971. The use of hierarchic clustering in information retrieval. *Information storage and retrieval* 7, 5 (1971).

[12] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proc. SIGIR*.

[13] Oren Kurland. 2006. *Inter-Document Similiarities, Language Models, and Ad Hoc Information Retrieval*. Ph. D. Dissertation.

[14] Oren Kurland. 2013. The Cluster Hypothesis in Information Retrieval. In *Proc. SIGIR*.

[15] Oren Kurland and Carmel Domshlak. 2008. A Rank-Aggregation Approach to Searching for Optimal Query-Specific Clusters. In *Proc. SIGIR*.

[16] Oren Kurland and Lillian Lee. 2004. Corpus Structure, Language Models, and Ad Hoc Information Retrieval. In *Proc. SIGIR*.

[17] Anton Leuski. 2001. Evaluating Document Clustering for Interactive Information Retrieval. In *Proc. CIKM*.

[18] J. Lin, R. Nogueira, and A. Yates. 2021. *Pretrained Transformers for Text Ranking: BERT and Beyond*. Morgan & Claypool Publishers.

[19] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2021. In-Batch Negatives for Knowledge Distillation with Tightly-Coupled Teachers for Dense Retrieval. In *Proc. RepL4NLP-2021*.

[20] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. 2020. Efficient Document Re-Ranking for Transformers by Precomputing Term Representations. In *Proc. of SIGIR*.

[21] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. 2020. Expansion via Prediction of Importance with Contextualization. In *Proc. of SIGIR*.

[22] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized Embeddings for Document Ranking. In *Proc. SIGIR*.

[23] Craig Macdonald, Rodrygo L. T. Santos, and Iadh Ounis. 2012. The whens and hows of learning to rank for web search. *Information Retrieval* 16 (2012).

[24] Craig Macdonald, Nicola Tonellotto, Sean MacAvaney, and Iadh Ounis. 2021. PyTerrier: Declarative Experimentation in Python from BM25 to Dense Retrieval. In *Proc. of CIKM*.

[25] Joel Mackenzie, Matthias Petri, and Alistair Moffat. 2021. A Sensitivity Analysis of the MSMARCO Passage Collection. *arXiv preprint 2112:03396* (2021).

[26] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning Passage Impacts for Inverted Indexes. In *Proc. SIGIR*.

[27] Antonio Mallia, Michal Siedlaczek, Joel Mackenzie, and Torsten Suel. 2019. PISA: Performant Indexes and Search for Academia. In *Proc. OSIRRC@SIGIR*.

[28] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint 1901.04085* (2019).

[29] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document Ranking with a Pretrained Sequence-to-Sequence Model. In *Proc. EMNLP*.

[30] Rodrigo Nogueira and Jimmy Lin. 2019. From doc2query to docTTTTTquery. *Online preprint* (2019).

[31] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-stage document ranking with BERT. *arXiv preprint 1910.14424* (2019).

[32] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021. The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. *arXiv preprint 2101:05667* (2021).

[33] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020).

[34] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (2009).

[35] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2013. Efficient and Effective Retrieval Using Selective Pruning. In *Proc. WSDM*.

[36] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2018. Efficient Query Processing for Scalable Web Search. *Found. Trends Inf. Ret.* 12, 4–5 (2018).

[37] Ellen M. Voorhees. 1985. The Cluster Hypothesis Revisited. In *Proc. SIGIR*.

[38] Xiao Wang, Sean MacAvaney, Craig Macdonald, and Iadh Ounis. 2022. An Inspection of the Reproducibility and Replicability of TCT-ColBERT. In *Proc. SIGIR*.

[39] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2021. Pseudo-Relevance Feedback for Multiple Representation Dense Retrieval. In *Proc. ICTIR*.

[40] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *Proc. ICLR*.

[41] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. Improving Recommendation Lists through Topic Diversification. In *Proc. WWW*.