End-to-End Query Term Weighting

Karan Samel[*]
Georgia Tech
ksamel@gatech.edu

Cheng Li
Google
chgli@google.com

Weize Kong
Google
weize@google.com

Tao Chen
Google
taochen@google.com

Mingyang Zhang
Google
mingyang@google.com

Shaleen Gupta
Google
shaleeng@google.com

Swaraj Khadanga
Google
khadanga@google.com

Wensong Xu
Google
asong@google.com

Xingyu Wang
Google
xingyuwang@google.com

Kashyap Kolipaka
Google
kashyapk@google.com

Michael Bendersky
Google
bemike@google.com

Marc Najork
Google
najork@google.com

ABSTRACT

Bag-of-words based lexical retrieval systems are still the most commonly used methods for real-world search applications. Recently deep learning methods have shown promising results to improve this retrieval performance but are expensive to run in an online fashion, non-trivial to integrate into existing production systems, and might not generalize well in out-of-domain retrieval scenarios. Instead, we build on top of lexical retrievers by proposing a Term Weighting BERT (TW-BERT) model. TW-BERT learns to predict the weight for individual n-gram (e.g., uni-grams and bi-grams) query input terms. These inferred weights and terms can be used directly by a retrieval system to perform a query search. To optimize these term weights, TW-BERT incorporates the scoring function used by the search engine, such as BM25, to score query-document pairs. Given sample query-document pairs we can compute a ranking loss over these matching scores, optimizing the learned query term weights in an end-to-end fashion. Aligning TW-BERT with search engine scorers minimizes the changes needed to integrate it into existing production applications, whereas existing deep learning based search methods would require further infrastructure optimization and hardware requirements. The learned weights can be easily utilized by standard lexical retrievers and by other retrieval techniques such as query expansion. We show that TW-BERT improves retrieval performance over strong term weighting baselines within MSMARCO and in out-of-domain retrieval on TREC datasets.

[*]Work done during an internship at Google.

KDD '23, August 6–10, 2023, Long Beach, CA, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599815

CCS CONCEPTS

• **Information systems → Language models**; **Query representation**.

KEYWORDS

Information Retrieval, Query Weighting, Language Models

ACM Reference Format:
Karan Samel, Cheng Li, Weize Kong, Tao Chen, Mingyang Zhang, Shaleen Gupta, Swaraj Khadanga, Wensong Xu, Xingyu Wang, Kashyap Kolipaka, Michael Bendersky, and Marc Najork. 2023. End-to-End Query Term Weighting. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3580305.3599815

1 INTRODUCTION

Lexical retrievers are the most commonly used information retrieval (IR) systems used in production applications. These systems have been built and refined over many decades to improve heuristic-based search using word statistics occurring within the search query, candidate document, and document corpus. These statistics-based retrieval methods provide efficient search that scales up with the corpus size and generalizes to new domains. However, the terms are weighted independently and don't consider the context of the entire query. For this problem, deep learning models can perform this contextualization over the query to provide better representations for individual terms. We bridge these two paradigms to determine which are the most relevant or non-relevant search terms in the query, which can be n-grams spanning multiple words. Then these terms can be up-weighted or down-weighted to allow our retrieval system to produce more relevant results.

A motivating search query is "Nike running shoes". The first aspect to consider is how these terms will be weighted during scoring. For example, the term "running" might be up-weighted due to its term frequency in the document corpus. This might provide other brand results while a customer intends to retain the "Nike" brand. Therefore the challenge is that we must ensure that

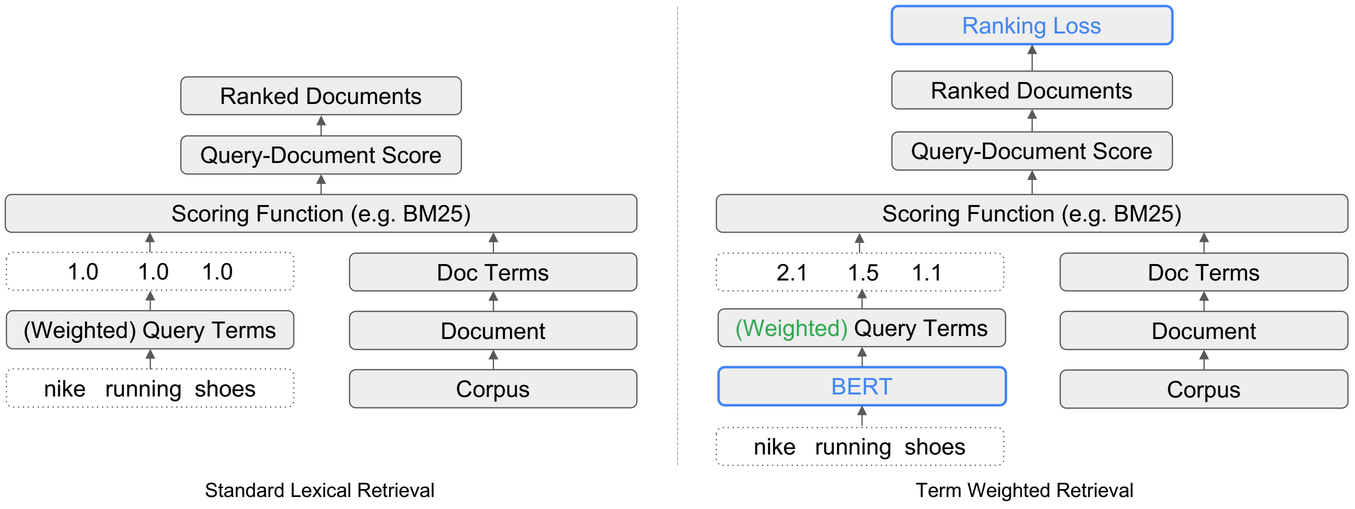


Figure 1: On the left we illustrate the traditional IR setup, which takes in a query with optional term weights (uniform by default) and scores them against documents from a corpus. In our setup on the right, we insert a BERT model to perform the weighting of the terms and train this model end-to-end by computing a ranking loss using the scored document positions.

"Nike" is weighted high enough while still providing running shoes in the final returned results.

The second aspect is how to leverage more meaningful n-gram terms during scoring. In our query, the terms "running" and "shoes" are handled independently, which can equally match "running socks" or "skate shoes". In this case, we want our retriever to work on an n-gram term level to indicate that "running shoes" should be up-weighted when scoring.

The challenge of n-gram weighting has traditionally been addressed by token-level term dependency and proximity methods, which identify salient n-grams to use for retrieval [5–7, 38]. However, using bag-of-words representations limits variations in the query, such as "running shoe" versus "joggers". Moreover, they require computing the term dependency statistics on any datasets used for testing, which limits their zero-shot capabilities. In contrast, term expansion methods leverage the training corpus to directly optimize term weights using auxiliary scoring functions [4, 8, 9]. However, these auxiliary scoring functions do not account for additional weighting steps carried out by scoring functions used in existing retrievers, such as query statistics, document statistics, and hyperparameter values. This can alter the original distribution of assigned term weights during final scoring and retrieval.

Beyond token-based statistics, deep learning models learn token and text-level representations to optimize retrieval given a set of queries and relevant documents. One class of models represents the text in a dense fashion to match similar query and document embeddings [17, 21, 26]. A second class of models sparsely output term wordpiece weights and corresponding term expansions to match the overlapping query and document terms [2, 15, 16, 16, 42]. These deep methods use pre-trained language models [13, 23, 40] as text encoders, and have shown large improvements in retrieval performance over traditional retrievers.

These gains from deep learning based search come with a few trade-offs. The first consideration is the complexity of these models, which incur challenges when deploying in a production use case. These models have a large inference cost, requiring specialized acceleration hardware and software [20, 32] to reduce latency. This provides additional overhead when integrating these methods into existing production systems. A second consideration is that most deep models are black box. The result of this is unpredictable behavior when testing on new domains whose data varies from the original training data [10, 39]. Correspondingly, a third consideration is the difficulty to program behaviors befitting new production use cases. Traditional retrievers in production can be adapted by providing new scoring functions based on new search heuristics. In contrast, implementing a similar behavior into deep retrieval is non-trivial.

To bridge the gap, we leverage the robustness of existing lexical retrievers with the contextual text representations provided by deep models. Lexical retrievers already provide the capability to assign weights to query n-gram terms when performing retrieval. We leverage a language model at this stage of the pipeline to provide appropriate weights to the query n-gram terms. This Term Weighting BERT (TW-BERT) is optimized end-to-end using the same scoring functions used within the retrieval pipeline to ensure consistency between training and retrieval. This leads to retrieval improvements when using the TW-BERT produced term weights while keeping the IR infrastructure similar to its existing production counterpart. We illustrate this augmentation of the lexical retrieval pipeline in Figure 1.

Methodological Contributions. We define the TW-BERT model, which provides n-gram weights given an input sequence of query tokens. We describe how to provide a relevance score for a query-document pair. This is done via a scoring mechanism that leverages both the produced term weights as well as traditional term statistics used by retrieval scoring functions. These relevance scores can be used to train the query term weights end-to-end. This enables us

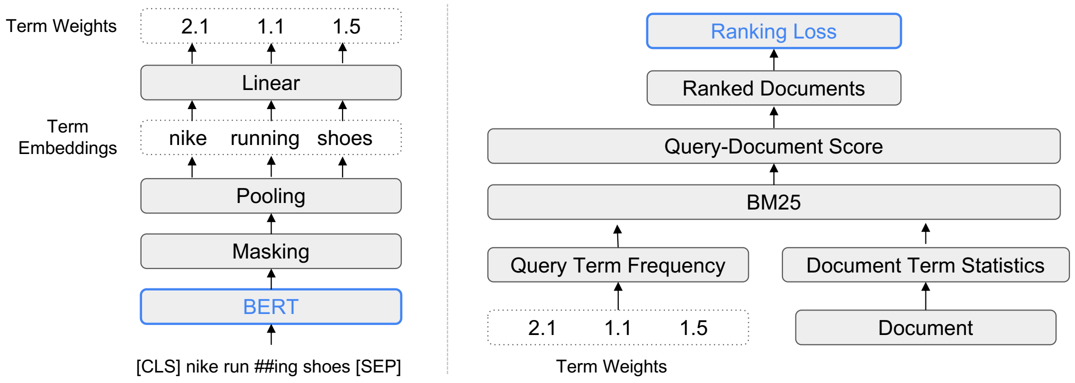


Figure 2: An overview of our TW-BERT framework. On the left, we illustrate the process of obtaining n-gram term weights from the original input wordpiece tokens. On the right, the term weights are used as statistics within our implemented scoring function, BM25 in this case, to score candidate documents and compute a final ranking loss.

to *directly* deploy our term weights within an IR system during retrieval. This differs from prior weighting methods which need to further tune a retriever's parameters to obtain optimal retrieval performance since they optimize term weights obtained by heuristics instead of optimizing end-to-end. We discuss TW-BERT's optimization design choices as well as the constraints that are needed to leverage the inference results within an external IR system. An overview of this process is presented in Figure 2.

Experimental Contributions. We leverage TW-BERT n-gram weights within an established lexical retriever platform to evaluate our system. We test our retrieval setup on in-domain tasks as well as in the zero-shot setting. TW-BERT weighting outperforms traditional retriever scoring for in-domain tasks while beating both traditional and deep retrievers on zero-shot retrieval. We additionally test TW-BERT weighting with query expansion methods available within our IR engine, which shows further performance improvements on zero-shot tasks.

2 RELATED WORKS

Traditional lexical retrieval uses scoring functions to rate query-document pairs. Methods such as TF-IDF or BM25 [36] are used to score the relevance of a document given query term statistics, document term statistics, and pre-computed term statistics from the entire document corpus. Relevance models have been used to compute the likelihood that a query term belongs to a set of relevant documents, without any explicit training labels [22]. Conversely, non-parametric language models are built per document where the probabilities of individual query terms can be inferred [33]. To weight n-gram spans, existing classical works focus on the interaction between the terms based on their text ordering and occurrences in n-gram spans given query-document pairs [5, 6, 38]. In the supervised case, probabilistic models maximize the scores for query terms given a related document [4, 9]. These in turn can score the likelihood of a query given a document for ranking. All aforementioned methods use co-occurrence probabilities at a *term* level to compute scores, while we would also benefit from using the *query* level information to weight the terms.

For this query contextualization, neural-based approaches DeepTR [43] and DeepCT [12] leverage word vectors [28] and BERT [13] encodings respectively to predict per term weights given the encoded query. They train the models to mimic term recall, which is how often a term appears across related queries associated with a relevant document. They can be similarly applied to weight and expand document-side terms [11, 27]. One drawback is that term recall is used as the ground truth weight for the model to mimic, which may not be the most optimal for downstream IR scoring. This is because the produced weight might not be compatible with the scoring function of the search engine. Moreover, there are a sparse number of queries associated with each document in most supervised datasets, thus term recall will often provide a noisy estimation of term weights. Instead of optimizing for an intermediate term weight heuristic, we aim to replicate IR scoring and optimize our term weights for end-to-end retrieval.

Beyond weight heuristic labels, other methods leverage pre-trained language models themselves to infer term weights. BERT-based models are used to compute independent query term scores given a candidate document [29]. These query scores can be added up to provide an overall query-document score. Transformer decoders can also take in the document to produce a probabilistic query, which in turn can be used to score query-document pairs [30, 31]. These methods produce more reliable similarity scores for each pairwise query document, but they are too expensive for retrieval.

Another promising line of work is to use sparse representations to encode the texts, which can be precomputed. Within sparse embeddings, SparTerm [2] uses the BERT masked language model (MLM) head to predict the importance of each term in the text. It learns to gate the term activations to enforce sparsity. Further extensions are made by SPLADE to normalize weight activations and improve sparse regularizations [16, 42]. These neural retrievers are improved upon by additional distillation and hard-negative mining techniques to generate higher quality training data [14, 15]. These methods pose challenges in integrating their wordpiece level terms, expansions, and document side weights into traditional retrievers. Therefore, we focus on optimizing word-level query term weights compatible with existing retrieval systems.

3 TERM WEIGHTING BERT

To efficiently integrate contextual text representations into IR systems, we design a BERT-based model to infer term weights for n-grams in queries. These n-gram term weights are used to construct a query representation that is used by the retrieval engine to perform document ranking. This query representation is passed through a standard retrieval scoring function, such as BM25, to rank candidate documents. A ranking loss is computed after the scoring function step, enabling the inferred TW-BERT term weights to be optimized end-to-end. This end-to-end training allows the term weights produced by TW-BERT to be used directly within existing IR systems. Therefore, it is more practical to integrate and deploy over pure deep learning approaches.

3.1 Model Architecture

We construct TW-BERT on top of the original BERT architecture but make modifications to support n-gram terms. In IR systems we want to provide weights for word-level terms, and not the wordpiece representations that are commonly used during tokenization. To this end, we define a term mask M that identifies which wordpiece token $W_i \in W$ belongs to each n-gram term in the query input $T_i \in T$. In the mask M, wordpieces belonging to n-gram terms are indicated with 1 or are 0 otherwise.

$$M_{i,j} \in \mathbb{R}^{|T|\times|W|} = \begin{cases} 1 & \text{if } W_j \text{ is a wordpiece of } T_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

An example of this mask can be viewed in Figure 3 where we demonstrate masking for uni-gram as well as bi-gram terms T.

Given the mask M, we can obtain each contextualized wordpiece token corresponding to each n-gram term in the query. We input the tokenized wordpiece query input to our BERT model and concatenate output wordpiece hidden states with dimension d as follows: $H = [h_1; h_2; h_3; \ldots; h_{|W|}] = \text{BERT}(W)$, where $H \in \mathbb{R}^{|W|\times d}$. Then the corresponding hidden states per n-gram term are computed and pooled to obtain a n-gram term level embedding for each input term. This is done by first expanding the dimensions of our embeddings $H^e \in \mathbb{R}^{1\times|W|\times d}$ as well as the mask matrix $M^e \in \mathbb{R}^{|T|\times|W|\times 1}$. Then the expanded wordpiece embeddings per n-gram term are computed as $E = H^e \odot M^e \in \mathbb{R}^{|T|\times|W|\times d}$. Here we are implicitly broadcasting during our $\odot$ multiplication, where for every n-gram term in T, we have all corresponding wordpiece embeddings $\in \mathbb{R}^{|W|\times|d|}$. Any non-relevant wordpiece embeddings per n-gram term are masked out as zero vectors. We pool the embeddings per n-gram over the wordpiece W dimension to obtain the term-level embeddings $P \in \mathbb{R}^{|T|\times d}$. In our use case, we compute the average pooled representations of the wordpieces as $P_{i,j} = \sum_{k=1}^{|W|} \frac{E_{i,k,j}}{M_{i,k}}$.

Given the n-gram term representations, we can predict their corresponding weight. This is done by a linear layer to obtain predicted weights $w = PK^\top + l$, where $w \in \mathbb{R}^{|T|}, K \in \mathbb{R}^{1\times d}$, and $l \in \mathbb{R}^{|T|}$. Since term weights should be non-negative, we use a ReLU activation to produce the final weight $w_i = \max(0, w_i)$. A summary of this term weighting process is shown on the left of Figure 2.

3.2 Retrieval Scoring

We integrate the inferred query term weights learned into our retriever by passing them into a corresponding retrieval scoring function. In our study, we use the BM25 scoring function. Given the query terms T, document terms D, and query term weights w this score is computed as follows:

$$\text{score}(T, D, w) = \sum_{i=1}^{|T|} \text{IDF}(T_i) \cdot \frac{f(T_i, D) * (k_3 + 1) * f(T_i, T, w)}{(k_3 + f(T_i, T, w)) * K} \tag{2}$$

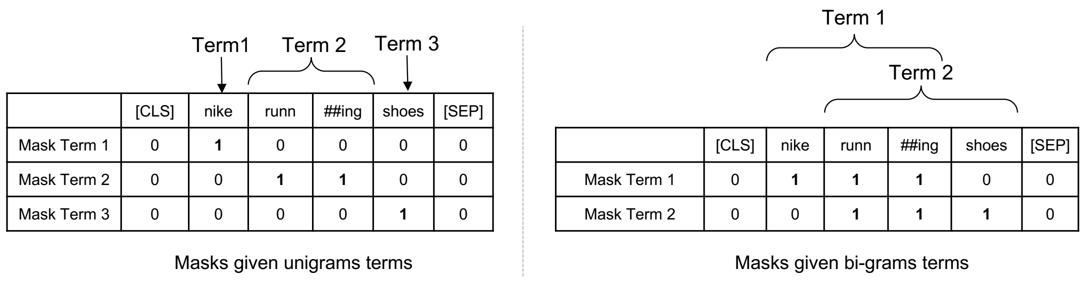
Here the inverse document frequency of the query term is computed as $\text{IDF}(T_i) = \ln(\frac{\text{numDocs}-\text{DF}(T_i)+0.5}{\text{DF}(T_i)+0.5})$, where DF is the number of documents in the corpus containing T_i. The document side frequency $f(T_i, D)$ is the number of times the term appears in the document. The query side frequency $f(T_i, T, w)$ is the weighted count of the term occurrence in the query. Using our predicted term weights w, this is computed as $f(T_i, T, w) = \sum_{j=1}^{|T|} \mathbb{I}_{T_i = T_j} * w_j$. Finally, K is another parametric constant defined as $K = k_1 * ((1 - b) + b * \frac{|D|}{ave|D|}) + f(T_i, D)$. We initialize the remainder scoring hyperparameters as $k_1 = 1.2, k_3 = 8.0$, and $b = 0.75$, which are the defaults used by the search engine we experiment with. Now given an input query and a corpus document, we can produce a matching score $s = score(T, D, w)$ using intermediate term weights w.

One benefit of modeling our scoring function is we are optimizing around our defined hyperparameter constants used by the search engine. In contrast, other weight learning methods like DeepTR [43] or DeepCT [12] still have to search over these hyperparameters to make sure that their learned weights produce viable results. Additionally, while we use BM25 as our scoring function, any generic scoring function can be used. These scorers can be newly developed for a specialized application or transferred from existing scorers used in production applications.

3.3 Scoring Regularization

We empirically find that directly optimizing these scoring functions is challenging. The first challenge is that the output scores fall in the range of $s \in [0, \infty)$, while query-document relevance labels (binary or graded relevance) range from $[0, 1]$. To address this we sample document scores s_c from a standard retrieval system and normalize the model output score ranges based on maximum observed document scores. We can define our scaled scores as $s_l = (s + \epsilon) * \frac{1}{\max s_c}$.

The second challenge we encounter is that custom or more complex scoring functions beyond BM25, which can be piecewise functions, are harder to optimize directly. For this, we optimize a piecewise linear transformation on top of the document scores in an end-to-end manner by applying a maxout network [18]. This network is added on top of the scaled scores as $s_m = \max_{i\in[1,L]} s_l a_i + b_i$, where L is the number of linear layers we use in the maxout computation and each $a_i, b_i \in \mathbb{R}$ are a unique set of parameters. In practice, we keep the $L = 12$ as the same number of transformer layers. These s_m scores are the final scores used for training and evaluation purposes.



	[CLS]	nike	runn	##ing	shoes	[SEP]
Mask Term 1	0	**1**	0	0	0	0
Mask Term 2	0	0	**1**	**1**	0	0
Mask Term 3	0	0	0	0	**1**	0

	[CLS]	nike	runn	##ing	shoes	[SEP]
Mask Term 1	0	**1**	**1**	**1**	0	0
Mask Term 2	0	0	**1**	**1**	**1**	0

Figure 3: For TW-BERT masking, we illustrate which wordpiece tokens (columns) belong to which n-gram tokens. On the left side, we have an example with uni-grams and on the right, we show a sample with bi-gram terms.

3.4 Optimization

3.4.1 Pre-Training. During our model pre-training, one objective is to run a masked language model (MLM) task [13]. Here each wordpiece has a 15% chance of being selected, of which it then has either an 80% chance of being masked, 10% chance of being randomly replaced, or 10% chance of remaining unchanged. In addition, we modify the masking strategy such that if any wordpiece is masked, all occurrences of that wordpiece are also masked. This makes the MLM task harder. The objective is to recover any selected token within the MLM loss $\mathcal{L}_{MLM}$.

The second pre-training objective is to initialize the term scores. We use an uninformed prior to mimic the standard retrieval scoring where the input query term weights are the same, as previously shown on the left of Figure 1. We pre-train the model to learn uniform weights by minimizing an MSE loss over the originally predicted term weights $\mathcal{L}_{MSE} = \sum_{j=1}^{|T|}(w_j - 1)^2$. The intuition is that we want to initialize the model to behave as a vanilla retriever system without any prior weight information during pre-training. Then during fine-tuning, we further optimize the weights to improve upon the standard uniform weighting strategy. During pre-training, we only require sample queries and do not require relevant documents. Our final joint pre-training objective is defined as $\mathcal{L}_{pre-train} = \mathcal{L}_{MLM} + \mathcal{L}_{MSE}$

3.4.2 Fine-Tuning. During fine-tuning, we leverage supervised query and relevant document pairs. During the forward pass, the input query is fed into TW-BERT, to produce the term weights. The document statistics are extracted for the document terms. These weights and statistics are fed into our BM25 scoring function and regularization layers to produce a final score s_m. This score is compared against the ground truth relevance of the query-document pair $y \in [0, 1]$. Due to the previously discussed optimization complexity with weighting and scoring, we further stabilize the optimization procedure by reserving gradient updates for large weight updates. This is done using an adapted MSE loss $\mathcal{L}_{AMSE}$ to score the prediction:

$$\mathcal{L}_{AMSE}(s_m, y) = \begin{cases} 0 & \text{for } |s_m - y| < d_1 \\ \frac{1}{2}(s_m - y)^2 & \text{for } d_1 \le |s_m - y| < d_2 \\ d_2 * (|s_m - y| - \frac{1}{2}d_2) & \text{otherwise} \end{cases} \tag{3}$$

Here $d_1 < d_2$ are hyperparameters, which we set to $d_1 = 0.2, d_2 = 1$. The first loss term states that if the predicted value is close to the provided label, then don't add any additional loss. The second term is the standard MSE loss. The third term is a Huber Loss [19], which reduces the penalty for large errors. These components of the loss let the model stay close to the uniform prior term weights while making small adjustments to optimize over the training dataset. This loss is also more robust when training on larger real-world datasets by avoiding updates caused by noise in labels.

Since retrieval is evaluated by the ranking of served documents, we also incorporate a list-wise ranking loss. Before training, we sample a set of negative documents for each query, where we use a T5 model [35] to score candidate query-document pairs. This is done by computing the likelihood of each query given the sample document. For each query, we use max normalization to produce weak labels $\in [0, 1]$ for corresponding documents, where we kept the top 1000 scoring documents per query. During training, we sample documents and labels given a single fixed query for each batch. Then given the model's output score for each document, we compute a ListMLE loss [41] $\mathcal{L}_{LMLE}$ as an additional signal to correctly order the scores. Our final fine-tuning objective is $\mathcal{L}_{fine-tune} = \mathcal{L}_{AMSE} + \mathcal{L}_{LMLE}$.

3.5 Methodological Advantage

We describe our TW-BERT architecture that inputs a text query and predicts query term weights for any size n-grams. All term weights are predicted in a single forward pass, which makes it tractable to perform during serving. Since we have a weight associated with each term, it also makes our method interpretable. These term weights are used by the same scoring functions used in IR engines to rank query-document pairs. We describe how to optimize this scoring effectively end-to-end, allowing us to leverage TW-BERT produced weights directly within IR engines for final retrieval.

4 EXPERIMENTAL SETUP

In our evaluation, we first describe the setup of our retriever system. Given the retriever setup, we denote the datasets, preprocessing, and tasks we evaluate on. Finally, we list the relevant baselines for our work.

4.1 Retriever Setup

We aim to leverage a lexical retrieval framework similar to those used in production settings for our evaluations. The motivation is that if we can integrate TW-BERT within such systems, our method can be more easily applied to existing production search applications. For our experiments, we use the Terrier IR Platform[1], which ingests weighted query terms and returns a ranked list of indexed documents. For the Terrier settings, we use the default processing pipeline which includes stopword removal and a Porter stemmer [34]. Note that similar pre-processing is done for the queries during TW-BERT training. We use the corresponding BM25 scorer in Terrier as well with the same hyperparameter settings. During our experiments, we evaluate the ranked documents directly returned by Terrier.

The main change from a vanilla Terrier deployment is that we add n-gram query term weights provided by TW-BERT. For the query, we use a max n-gram term size of 2 for all our experiments. This means that for each candidate query, we utilize all its uni-gram as well as all bi-gram terms. In the input sequence, we concatenate the uni-grams followed by the bi-gram terms. Recall that each uni-gram and bi-gram have distinct embeddings since we mask each corresponding wordpiece token to each distinct n-gram term. TW-BERT provides a weight for each of these uni-grams and bi-grams occurring in the query. The corresponding n-grams and their weights are then formatted using the Indri query language [37]. This formatted query is directly used by the Terrier pipeline to retrieve the ranked documents.

4.2 Baseline Methods

We also test other baseline methods to weigh our n-gram query terms. Note that for all methods we format the terms and weights into an Indri query, keeping the same Terrier evaluation setup as TW-BERT weighting. In particular, we test different approaches to weight the input query terms, while the Terrier BM25 scoring function and the retrieval pipeline remain constant.

We first compare the most standard BM25 retrieval setup, where the input term weights passed into the scoring function are uniform. Another input term weighting method we test is DeepCT [12] where the model optimizes to mimic query term recall. Unlike the original paper, which applies the weights to the document side, we apply the inferred weights to the query terms only. We also test SPLADE [16], where we use the query side model to produce the query term weights. However, SPLADE produces weights at a wordpiece level. To operate at our n-gram level, we average pool the wordpiece weights corresponding to each n-gram term. We focus on term weighting methods from these deep pre-trained models to allow for testing on out-of-domain terms, while classical weighting methods [5–7, 38] use in-domain term statistics.

Finally, we test a Neural Passage Retrieval (NPR) model [25], which has two dense BERT encoders to encode the query and the document passages. It uses an embedding dot product to rank query-document pairs. This baseline is the exception for our lexical retrieval evaluation, as it does not use Terrier. It is used instead to provide a reference point to compare dense models against augmented lexical retrievers, regardless of deployment constraints.

[1]http://terrier.org

Table 1: Statistics regarding our evaluation datasets.

Dataset	Corpus Size	# Test Queries
MSMARCO	8.8M	101k
TREC-COVID	191k	50
Robust04	528k	250
Common Core	728k	50

Table 2: MSMARCO dev evaluation results which are run through Terrier using different input term weighting strategies.

Input Weights	mAP	R@10	R@100	R@500	R@1000
BM25	.1931	.3856	.6739	.8249	.8714
DeepCT (Q)	.1932	.3862	.6737	.8247	.8713
SPLADE (Q)	.1923	.3858	.6805	**.8314**	**.8760**
TW-BERT	**.2025**	**.4091**	**.6949**	.8295	.8678
NPR	.3547	-	-	-	.9795

4.3 Datasets and Tasks

We pre-train and fine-tune all models on our T5 augmented MSMARCO passage data [3]. In both stages, we only use the train split for optimization and use the dev set for evaluation.

We further split the MSMARCO train set into 95% training and 5% validation sets. During pre-training we use the published pre-trained BERT-base checkpoint[2]. We further run our pre-training MLM and weight initialization tasks on top of the pre-trained checkpoint. We run our fine-tuning losses on top of this second-stage pre-trained model. For both training stages, we use the validation split set to perform early stopping. We use an AdamW optimizer [24] with a learning rate of $1e^{-5}$ for both stages.

In addition to the MSMARCO evaluation, we also evaluated our performance on three other datasets:

(1) TREC-COVID[3]: COVID related articles and studies.
(2) Robust04[4]: News article datasets.
(3) Common Core[5]: Educational articles and blog posts.

For these datasets, we only have a small number of evaluation queries and we do not fine-tune any of our models on these datasets. Therefore these tasks are zero-shot attempts by our models. The corpus and query size statistics for these datasets are shown in Table 1.

5 EXPERIMENTAL RESULTS

5.1 In-domain Results

We compare TW-BERT against our baseline weighting strategies when evaluating the MSMARCO dev set in Table 2. Here the uniform term weighting strategy (BM25) is the only baseline that is not supervised, while the remainder models are optimized on the training split. We explicitly denote that for SPLADE (Q) and DeepCT (Q),

[2]https://huggingface.co/bert-base-uncased
[3]https://ir.nist.gov/trec-covid/
[4]https://ir-datasets.com/trec-robust04.html
[5]http://trec.nist.gov/data/wapost/

Table 3: Zero-shot results for TREC-COVID, Robust04, and Common Core. The top section shows results of our query weighting baselines in Terrier. The middle sections shows the uniform weighting and TW-BERT results when using Bo1 expansions. The bottom row shows the dense retrieval results using the NPR model.

		TREC-COVID		Robust04		Common Core	
Input Weights	Scoring	mAP	R@1000	mAP	R@1000	mAP	R@1000
BM25	BM25	.2026	.3954	.2442	.7007	.2159	.6927
DeepCT (Q)	BM25	.2025	.3952	.2443	**.7008**	.2164	.6906
SPLADE (Q)	BM25	.1977	.3898	.2337	.6882	.2082	.6925
TW-BERT	BM25	**.2207**	**.4126**	**.2447**	.6812	**.2412**	**.7059**
BM25	Bo1 Exp. + BM25	.2266	.4273	.2767	**.7581**	.2950	.7645
TW-BERT	Bo1 Exp. + BM25	**.2328**	**.4305**	**.2823**	.7542	**.3213**	**.7837**
NPR	Dense Retrieval	.1107	.2523	.2540	.6743	.2557	.7124

Table 4: Stratified MSMARCO dev mAP results based on the length of the query. The query length is dictated by the number of uni-grams in the query.

Input Weights	Length 1-5	Length 6-10	Length 11+
BM25	.2036	.1816	**.1920**
TW-BERT	**.2133**	**.1919**	.1915
% Data	42.7	50.9	7.4

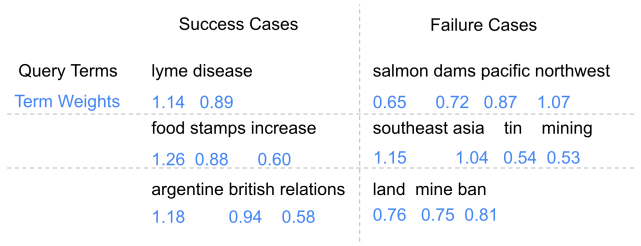


Figure 4: Sample inference results from TW-BERT on Robust04 queries. We show the term weights that led to wins on the left and losses on the right respectively against a uniform weighting strategy.

we only use the query model and apply the weights to the original query terms. This differs from their original setups that modified document side weights as well.

From the methods that utilize Terrier for retrieval, TW-BERT has the best balance in mAP over precision and recall. Furthermore, we see that it can retrieve the most relevant results in the top 100 documents. We also observe that TW-BERT obtains the largest gains in shorter queries, as shown in Table 4. In shorter queries, each weight provides stronger feedback over the retrieval of relevant documents. In longer queries, additional weighted terms can lead to noise and mask the effect of the salient terms.

We expect TW-BERT to improve over uniform weighting for in-domain tasks. After pre-training, TW-BERT outputs similar uniform weights but optimizes these term weights during the fine-tuning stage. For Deep-CT we see comparable performance to uniform. This is because in the training data, the number of queries associated with a document is sparse. Therefore it is difficult to estimate term recall, which usually ends up being close to uniform. Testing weighted term recall based on T5 labeled pairs led to similar performance as well. SPLADE had on-par mAP to the baselines but showed improvements in long-tail recall. One factor in this is due to the MLM pre-training, which leads to robust wordpiece weights even if the entire word-level term is not seen in the training set.

The pure neural approach NPR performs the best overall when operating on the in-domain test set. Unlike our Terrier baselines it learns a dense embedding for the entire query and document text to leverage for scoring. In contrast, our baselines only modify term weights and are restricted to a pre-defined scoring method and pre-computed term-level statistics. Unlike our IR results, this method is harder to integrate into existing IR infrastructure for deployment. These models also provide varying results when evaluated on out-of-domain data, which we present next.

5.2 Zero-shot Results

We observe the generalization capabilities of our models when inferring on out-of-distribution datasets, shown in Table 3. We first focus on retrieval results in the top section of the Table as well as the NPR results at the bottom row.

TREC-COVID. In TREC-COVID, TW-BERT still provides large gains over the baseline methods. One aspect of this dataset is that the COVID vocabulary is not present in the MSMARCO corpus. The metric improvements indicate that our model learns to disambiguate main topics from filler terms and weights them appropriately. The remainder weighting strategies provide un-informed weights per term and perform similarly to uniform weighting. We see the clear effect of out-of-distribution COVID vocabulary where NPR performs significantly lower than our retrieval methods.

Robust04. In Robust04, we see comparable results across all methods. Here standard retriever methods provide the best recall, while NPR retains precision-recall balance. In this data we observe a variety of proper nouns and unique entities not observed in MSMARCO, leading to similar performance across retrievers.

For Robust04 we also show sample term weighting by TW-BERT in Figure 4. Here we see examples when it correctly provides more relevant documents over uniform weighting on the left. For example, we identify in “lyme disease” that we need to indicate the “lyme” is the specific disease we want. Similarly, we want documents that correspond to “food stamps” and “argentine british” as opposed to just “increases” and “relations” in the 2nd and 3rd examples respectively. On the right, we also observe when our weighting method fails and uniform weighting provides more relevant results. In the first two examples, we see that our model focuses on the region of the query “pacific northwest” and “southeast asia”, rather than the topics of “salmon dams” or “tin mining”. Similarly, in the last example, we see a focus on “ban”, which can pertain to any ban rather than the “land mine” ban.

Common Core. In comparison to Robust04, Common Core contains more common terms overlapping with MSMARCO and therefore we see larger improvements for TW-BERT over other weighting methods. Since the input distribution is similar to our training data, we see that NPR also excels in this domain setup.

5.3 Query Expansion

Typically most retrievers are configured with query expansion capabilities, that include common co-occurring terms or synonyms of query terms. We test TW-BERT performance in this setting, by leveraging a Bo1 expansion model [1] provided by the Terrier platform. This involves running a first-pass retrieval of the query using uniform weights to generate candidate documents. This expansion model selects the top terms that explain the divergence between the top documents and a random distribution. In our setup, we use 10 terms and the top 5 documents. Then in the second pass, our TW-BERT model provides weights for the original and expanded terms, both of which are used for final retrieval. These expansion results are reported in the middle section of Table 3. We see even larger gains compared to uniform weighting expansion, even though our TW-BERT model was not trained using expansion term data. Additionally, we see that both expansion schemes provide significant improvements over NPR, reiterating the necessity of using lexical-based methods for out-of-domain tasks.

We note that we also tested SPLADE expansion by adding its pooled expansion terms to the input query. In these experiments, we found out that expansion performance was poorer than the non-expanded setup. This tells us SPLADE must match query terms against the weighted and expanded document side terms, which are non-trivial to integrate into the existing retriever pipeline. It also shows the importance to leverage term-level, rather than wordpiece level, representations for matching to integrate into lexical retriever systems.

In our overall zero-shot results, we see TW-BERT providing consistent improvements over term weighting baselines using the same retrieval pipeline. This improvement is consistent when using both the original terms and expanded terms. In comparison, we observe dense retrieval performance varies based on the term overlap with MSMARCO data.

6 CONCLUSION AND FUTURE WORK

Lexical retriever systems are still commonplace for large-scale search infrastructure. They leverage term statistics and heuristic scoring functions which enable efficient search, but may not capture the intent of the text. Recent deep learning approaches have led to better contextualization of input texts, but are non-trivial to deploy into existing search production systems. To bridge the gap we propose TW-BERT, which learns to weight search query input terms to identify salient inputs. Like traditional search, these weighted terms are fed into a retriever scoring function to compute the relevance of a candidate document. We compute a ranking loss from the scoring function outputs, which allows us to optimize our predicted term weights in an end-to-end fashion. With this end-to-end optimization, we can directly leverage our predicted weights within established lexical retriever frameworks without performing additional parameter tuning. Using these retriever frameworks, we show that our term weighting method outperforms baseline term weighting strategies for in-domain tasks. In out-of-domain tasks, TW-BERT improves over baseline weighting strategies as well as dense neural rankers. We further show the utility of our model by integrating it with existing query expansion models, which improves performance over standard search and dense retrieval in the zero-shot cases. This motivates that our work can provide improvements to existing retrieval systems with minimal onboarding friction. It also provides avenues for future improvements to our framework.

In our current work, we modify parts of an IR system that are flexible by default, such as the input search query weights and n-gram terms. In future work, we can soften these constraints to further improve performance. One direction is to consider weighing the document side terms as well. Similarly, if both query and document sides were trained with expansion terms, we can perform finer-grained matching. This can be seen as SPLADE but from the perspective of retaining IR scoring functions and operating on term-level tokens. Another aspect to investigate is out-of-domain retrieval performance. In particular, our model is trained with the corpus statistics of the original training corpus. These corpus statistics will vary when evaluating on different corpora. Therefore, building in robustness to how we leverage the original corpus statistics during scoring could further improve our method's generalization performance.

REFERENCES

[1] Giambattista Amati. 2003. *Probability models for information retrieval based on divergence from randomness Ph. D.* Ph.D. Dissertation. thesis. University of Glasgow.

[2] Yang Bai, Xiaoguang Li, Gang Wang, Chaoliang Zhang, Lifeng Shang, Jun Xu, Zhaowei Wang, Fangshan Wang, and Qun Liu. 2020. SparTerm: Learning term-based sparse representation for fast text retrieval. *arXiv preprint arXiv:2010.00768* (2020).

[3] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268* (2016).

[4] Michael Bendersky and W Bruce Croft. 2008. Discovering key concepts in verbose queries. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval.* 491–498.

[5] Michael Bendersky and W Bruce Croft. 2012. Modeling higher-order term dependencies in information retrieval using query hypergraphs. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval.* 941–950.

[6] Michael Bendersky, Donald Metzler, and W Bruce Croft. 2010. Learning concept importance using a weighted dependence model. In *Proceedings of the third ACM international conference on Web search and data mining.* 31–40.

[7] Michael Bendersky, Donald Metzler, and W Bruce Croft. 2011. Parameterized concept weighting in verbose queries. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval.* 605–614.

[8] Michael Bendersky, Donald Metzler, and W Bruce Croft. 2012. Effective query formulation with multiple information sources. In *Proceedings of the fifth ACM international conference on Web search and data mining*. 443–452.

[9] Guihong Cao, Jian-Yun Nie, Jianfeng Gao, and Stephen Robertson. 2008. Selecting good expansion terms for pseudo-relevance feedback. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 243–250.

[10] Tao Chen, Mingyang Zhang, Jing Lu, Michael Bendersky, and Marc Najork. 2022. Out-of-domain semantics to the rescue! zero-shot hybrid retrieval models. In *Advances in Information Retrieval: 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10–14, 2022, Proceedings, Part I*. Springer, 95–110.

[11] Zhuyun Dai and Jamie Callan. 2020. Context-aware document term weighting for ad-hoc search. In *Proceedings of The Web Conference 2020*. 1897–1907.

[12] Zhuyun Dai and Jamie Callan. 2020. Context-aware term weighting for first stage passage retrieval. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 1533–1536.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[14] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE v2: Sparse lexical and expansion model for information retrieval. *arXiv preprint arXiv:2109.10086* (2021).

[15] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From distillation to hard negative sampling: Making sparse neural ir models more effective. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2353–2359.

[16] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2288–2292.

[17] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: Revisit exact lexical match in information retrieval with contextualized inverted list. *arXiv preprint arXiv:2104.07186* (2021).

[18] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Maxout networks. In *International conference on machine learning*. PMLR, 1319–1327.

[19] Peter J Huber. 1992. Robust estimation of a location parameter. *Breakthroughs in statistics: Methodology and distribution* (1992), 492–518.

[20] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[21] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 39–48.

[22] Victor Lavrenko and W Bruce Croft. 2017. Relevance-based language models. In *ACM SIGIR Forum*, Vol. 51. ACM New York, NY, USA, 260–267.

[23] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[24] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

[25] Jing Lu, Gustavo Hernandez Abrego, Ji Ma, Jianmo Ni, and Yinfei Yang. 2021. Multi-stage training with improved negative contrast for neural passage retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 6091–6103.

[26] Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2020. Sparse, Dense, and Attentional Representations for Text Retrieval. CoRR abs/2005.00181 (2020). *arXiv preprint arXiv:2005.00181* (2020).

[27] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. 2020. Expansion via prediction of importance with contextualization. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 1573–1576.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[29] Bhaskar Mitra, Corby Rosset, David Hawking, Nick Craswell, Fernando Diaz, and Emine Yilmaz. 2019. Incorporating query term independence assumption for efficient retrieval and ranking using deep neural networks. *arXiv preprint arXiv:1907.03693* (2019).

[30] Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. 2019. From doc2query to docTTTTTquery. *Online preprint* 6 (2019).

[31] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document expansion by query prediction. *arXiv preprint arXiv:1904.08375* (2019).

[32] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139* (2017).

[33] Jay M Ponte and W Bruce Croft. 2017. A language modeling approach to information retrieval. In *ACM SIGIR Forum*, Vol. 51. ACM New York, NY, USA, 202–208.

[34] Martin F Porter. 2001. Snowball: A language for stemming algorithms.

[35] 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. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[36] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[37] Trevor Strohman, Donald Metzler, Howard Turtle, and W Bruce Croft. 2005. Indri: A language model-based search engine for complex queries. In *Proceedings of the international conference on intelligent analysis*, Vol. 2. Washington, DC., 2–6.

[38] Krysta M Svore, Pallika H Kanani, and Nazan Khan. 2010. How good is a span of terms? Exploiting proximity to improve web retrieval. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. 154–161.

[39] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663* (2021).

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[41] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*. 1192–1199.

[42] Jheng-Hong Yang, Xueguang Ma, and Jimmy Lin. 2021. Sparsifying Sparse Representations for Passage Retrieval by Top-k Masking. *arXiv preprint arXiv:2112.09628* (2021).

[43] Guoqing Zheng and Jamie Callan. 2015. Learning to reweight terms with distributed representations. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 575–584.