A Unified Framework for Learned Sparse Retrieval

Thong Nguyen[1], Sean MacAvaney[2], and Andrew Yates[1]

[1] University of Amsterdam
[2] University of Glasgow
t.nguyen2@uva.nl

Abstract. Learned sparse retrieval (LSR) is a family of first-stage retrieval methods that are trained to generate sparse lexical representations of queries and documents for use with an inverted index. Many LSR methods have been recently introduced, with Splade models achieving state-of-the-art performance on MSMarco. Despite similarities in their model architectures, many LSR methods show substantial differences in effectiveness and efficiency. Differences in the experimental setups and configurations used make it difficult to compare the methods and derive insights. In this work, we analyze existing LSR methods and identify key components to establish an LSR framework that unifies all LSR methods under the same perspective. We then reproduce all prominent methods using a common codebase and re-train them in the same environment, which allows us to quantify how components of the framework affect effectiveness and efficiency. We find that (1) including document term weighting is most important for a method's effectiveness, (2) including query weighting has a small positive impact, and (3) document expansion and query expansion have a cancellation effect. As a result, we show how removing query expansion from a state-of-the-art model can reduce latency significantly while maintaining effectiveness on MSMarco and TripClick benchmarks. Our code is publicly available.[3]

Keywords: neural retrieval · learned sparse retrieval · lexical retrieval.

1 Introduction

Neural information retrieval has becoming increasingly common and effective with the introduction of transformers-based pre-trained language models [17]. Due to latency constraints, a pipeline is often split into two stages: first-stage retrieval and re-ranking. The former focuses on efficiently retrieving a set of candidates to re-rank, whereas the latter focuses on re-ranking using highly effective but inefficient methods. Neural first-stage retrieval approaches can be grouped into two categories: dense retrieval (e.g., [12, 13, 38]) and learned sparse retrieval (e.g., [7, 40, 44]). Learned sparse retrieval (LSR) methods transform an input text (i.e., a query or document) into sparse lexical vectors, with each dimension

[3] Code: https://github.com/thongnt99/learned-sparse-retrieval

containing a term score analogous to TF. The sparsity of these vectors allows LSR methods to leverage an inverted index. Compared with dense retrieval, LSR has several attractive properties. Each dimension in the learned sparse vectors is usually tied to a term in vocabulary, which facilitates transparency. We can, for example, examine biases encoded by models by looking at the generated terms. Furthermore, LSR methods can re-use the inverted indexing infrastructure built and optimized for traditional lexical methods over decades.

The idea of using neural methods to learn weights for sparse retrieval predates transformers [40, 42], but approaches' effectiveness with pre-BERT methods is limited. With the emergence of retrieval powered by transformer-based pre-trained language models [6, 17, 36], many LSR methods [5, 7, 8, 16, 20, 23, 41] have been introduced that leverage transformer architectures to substantially improve effectiveness. Among them, the Splade [7] family is a recent prominent approach that shows strong performance on the MSMarco [26] and BEIR benchmarks [35].

Despite their architectural similarities, different learned sparse retrieval methods exhibit very different behaviors regarding effectiveness and efficiency. The underlying reasons for these differences are often unclear.

In this work, we conceptually analyze existing LSR methods and identify key components in order to establish a comparative framework that unifies all methods under the same perspective. Under this framework, the key differences between existing LSR methods become apparent. We first reproduce methods' original results, before re-training and re-evaluating them in a common environment that leverages best practices from recent work, like the use of hard negatives. We then leverage this setting to study how key components influence a model's performance in terms of efficiency and effectiveness. We investigate the following research questions:

RQ1: Are the results from LSR papers reproducible?
This RQ aims to reproduce the results of all recent, prominent LSR methods in our codebase, consulting the configuration on the original papers and codes. We find that most of the methods can be reproduced with MRR comparable to the original work (or slightly higher).

RQ2: How do LSR methods perform with recent advanced training techniques?
Splade models [7] show impressive ranking scores on MSMarco. While these improvements could be due to architectural choices like incorporating query expansion, Splade also benefits from an advanced training process with mined hard negatives and distillation from cross-encoders. Our experiments show that with the same training as Splade, many older methods become significantly more effective. Most noticeably, the MRR@10 score of the older EPIC [20] model was boosted by 36% to become competitive with Splade.

RQ3: How does the choice of encoder architecture and regularization affect results?
The common training environment we use to answer RQ2 allows us to quantify the effect of various architectural decisions, such as expansion, weighting, and

regularization. We find that document weighting had the greatest impact on a system's effectiveness, while query weighting had a moderate impact, though query weighting improves latency by eliminating non-useful terms. Notably, we observed a cancellation effect between improvements from document and query expansion, indicating that query expansion is not necessary for a LSR system to perform well.

Our contributions are: (**1**) an conceptual framework that unifies all prominent LSR methods under the same view, (**2**) an analysis of how LSR components affect efficiency and effectiveness, which e.g. leads to a modification that reduces more than 74% retrieval latency while keeping the same SOTA effectiveness, and (**3**) implementations of all studied methods in the same codebase, including simple changes in Anserini [39] that make LSR indexing faster.

2 Learned sparse retrieval

Learned sparse retrieval (LSR) uses a query encoder f_Q and a document encoder f_D to project queries and documents to sparse vectors of vocabulary size: $w_q = f_Q(q) = w_q^1, w_q^2, \ldots, w_q^{|V|}$ and $w_d = f_D(d) = w_d^1, w_d^2, \ldots, w_d^{|V|}$. The score between a query a document is the dot product between their corresponding vectors: $sim(q,d) = \sum_{i=1}^{|V|} w_q^i w_d^i$. This formulation is closely connected to traditional sparse retrieval methods like BM25; indeed, BM25 [32,33] can be formulated as:

$$\begin{aligned}
\text{BM25}(q,d) &= \sum_{i=1}^{|q|} \text{IDF}(q_i) \times \frac{tf(q_i,d) \times (k_1+1)}{tf(q_i,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)} \\
&= \sum_{j=1}^{|V|} \underbrace{\mathbb{1}_{q(v_j)} \text{IDF}(v_j)}_{\text{query encoder}} \times \underbrace{\mathbb{1}_{d(v_j)} \frac{tf(v_j,d) \times (k_1+1)}{tf(v_j,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)}}_{\text{doc encoder}} \\
&= \sum_{j=1}^{|V|} f_Q(q)_j \times f_D(d)_j
\end{aligned}$$

With BM25 the IDF and TF components can be viewed as query/document term weights. LSR differs by using neural models, typically transformers, to predict term weights. LSR is compatible with many techniques from sparse retrieval, such as inverted indexing and accompanying query processing algorithms. However, differences in LSR weights can mean that existing query processing optimizations become much less helpful, motivating new optimizations [21, 22, 24].

2.1 Unified learned sparse retrieval framework

In this section, we introduce a conceptual framework consisting of three components (*sparse encoder*, *sparse regularizer*, *supervision*) that captures the key differences we observe between existing learned sparse retrieval methods. Later, we describe how LSR methods in the literature can be fit into this framework.

Name	Backbone	Head	Expansion	Weighting
BINARY	Transf. Tokenizer	-	No	No
MLP	Transf. Encoder	Linear(s)	No	Yes
expMLP	Transf. Encoder	Linear(s)	Yes	Yes
MLM	Transf. Encoder	MLM Head + Agg.	Yes	Yes
clsMLM	Transf. Encoder	MLM Head	Yes	Yes

Table 1: Encoder architectures. (Transf: Transformers)

Sparse (Lexical) Encoders. A sparse or lexical encoder encodes queries and passages into weight vectors of equal dimension. This is the main component that determines the effectiveness of a learned sparse retrieval method. There are three distinct characteristics that make sparse encoders different from dense encoders. The first and most straightforward difference is that sparse encoders produce sparse vectors (i.e., most term weights are zero). This sparsity is controlled by sparse regularizers, which we will discuss in the next section.

Second, dimensions in sparse weight vectors are usually tied to terms in a vocabulary that contains tens of thousands of terms. Therefore, the size of the vectors is large, equal to the size of the vocabulary; each dimension represents a term (typically a BERT word piece). On the contrary, (single-vector) dense retrieval methods produce condensed vectors (usually fewer than 1000 dimensions) that encode the semantics of the input text without a clear correspondence between terms and dimensions. Term-level dense retrieval methods like ColBERT [13] do preserve this correspondence.

The third distinction is that encoders in sparse retrieval only produce non-negative weights, whereas dense encoders have no such constraint. This constraint comes from the fact that sparse retrieval relies on software stacks (inverted indexing, query processing algorithms) built for traditional lexical search (e.g., BM25), where weights are always non-negative term frequencies.

Whether these differences lead to systematically different behavior between LSR and dense retrieval methods is an open question. Researchers have observed that LSR models and token-level dense models like ColBERT tend to generalize better than single-vector dense models on the BEIR benchmark [8,35]. There are also recent works proposing hybrid retrieval systems that combine the strength of both dense and sparse representations [3, 18, 19], which can bring benefits for both in-domain and out-of-domain effectiveness [19].

There are several variants of sparse encoders, which are typically built on a transformer-backbone [36] with additional head layer(s) on top. In Table 1, we summarize a list of common architectures of sparse encoders proposed in the literature. We use the following notation when describing these sparse encoder architectures: v_i denotes the i^{th} term in a vocabulary V; t_j denotes the j^{th} term in an input sequence t (either a query or document) of length L; h_j represents the contextualized embedding of t_j from a transformer encoder; e_i represents the transformer's input embedding of the v_i; $w_i(t)$ represents the weight of v_i in the context of t. The architectures include:

- **BINARY**: The BINARY encoder simply tokenizes the input into terms (word pieces) and considers the presence of terms in the input text. The binary encoder performs neither term expansion nor weighting:

$$w_i(t) = \max_{j=1..L} \mathbb{1}\big(v_i = t_j\big) \tag{1}$$

- **MLP**: This encoder uses a **M**ulti-**l**ayer **P**erceptron (usually one layer) on top of each contextualized embedding h_j produced by the transformer-backbone for each input term to generate the term's score. Only terms in the input receive a weight; the other terms are zero.

$$w_i(t) = \sum_{j=1...L} log\left(\mathbb{1}(v_i = t_j)\Big(ReLU(h_j W + b)\Big) + 1\right) \tag{2}$$

  where W and b are the weight and bias of the linear head. This MLP architecture focuses on term weighting.
- **expMLP**: This encoder adds a pre-processing step to expand the input with relevant terms before using a MLP encoder. The expansion terms can be selected from an external source/model (e.g., DocT5Query [27]).
- **MLM**: The MLM encoder aggregates term weights over the logits produced by BERT's **M**asked **L**anguage **M**odel head. The weight for each term in the vocabulary is generated as follows:

$$w_i(t) = q(t) log\left(1 + \max_{j=1...L} ReLU\Big(h_j^\intercal e_i + b_i\Big) g(t_j)\right). \tag{3}$$

  The ReLU function ensures non-negative weights and can be replaced with e.g. a Softplus, which has similar properties but is differentiable everywhere. The *log* normalization prevents some weights from getting too large. Term importance and passage quality scores are captured by $g(t_j)$ and $q(t)$, respectively. When present, the $g(t_j)$ and $q(t)$ functions can be modeled by an linear layer on top of contextualized embeddings of input tokens and the [CLS] token. Out of the three approaches using a MLM encoder, only one includes these functions. The choice of max aggregation and ReLU activation makes sparser representations and, at the same time, reduces training time as they disconnect the output from many paths in the computational graph.
- **clsMLM**: This is a simplified version of the MLM encoder that only takes the logits of the [CLS] token, which is at the position 0 of the sequence, as the output vector. Intuitively, this encoder squeezes the information of the whole sequence into a small [CLS] vector, which is then projected into an over-complete set of vocabulary bases:

$$w_i(t) = ReLU(h_0^\intercal e_i + b_i) \tag{4}$$

  where h_0 is the contextualized embedding of the CLS token.

These encoders are defined independent of input type (i.e., query or document). We can use a single shared encoder to encode both queries and documents or employ two separate encoders mixed-and-matched from the above list.

Sparse regularizers. Sparse regularizers control the sparsity of weight vectors, which is crucial for query processing efficiency. We describe three common regularization techniques used in learned sparse retrieval methods.

- **FLOPs**: The FLOPs regularizer [29], estimates the average number of floating-point operations needed to compute the dot product between two weight vectors by a smooth function. FLOPs is defined over a batch of N sparse representations as follows:

$$FLOPs = \sum_{i=1}^{|V|} \bar{a_i}^2 = \sum_{i=1}^{|V|} \Big(\frac{1}{N} \sum_{j=1}^{N} w_j^i \Big)^2 \tag{5}$$

  where $\bar{a_i}$ is the estimated activation probability of the i^{th} dimension. Intuitively, the FLOPS regularizer might lead to two side-effects: (1) it forces the weights to be small and (2) it encourages uniform activation probability across all dimension when the square sum is minimized.
- L_p **Norm**: The family of L_p norms has been commonly applied in machine learning to mitigate over-fitting. With LSR, L_p is applied to the output vector rather than to model weights. L_1 and L_2 are two widely used norms.
- **Top-K**: This is a simple pruning technique which only keeps the top-k highest weights and zeroes out the rest. This pruning can be applied at inference time as a post-processing step or at training time with the value of k decreasing over time [20].

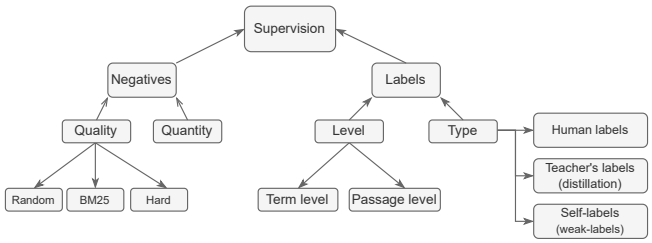

Fig. 1: Aspects of supervision commonly used for learned sparse retrieval.

Supervision. As some published LSR methods have identical sparse encoder(s) and sparse regularizer(s), we consider the supervision component to differentiate them and to consider its effect. As illustrated in Figure 1, this supervision component is composed of two factors: negative examples and labels.

- **Negatives:** For contrastive learning, the quality and number of negatives used for training have a significant impact on performance [1]. The more and harder the negatives, the better the result. A naive way of selecting negatives is randomly sampling non-positive passages/documents from the corpus [10, 20, 26, 44], but this tends to create easy, less informative examples. Harder negatives can be selected from the top non-positive documents returned by BM25 or by neural retrieval models [38], which can also be used to filter out false negatives [30].

- **Labels:** Labels for training LSR methods are classified by type and level. Types include human, teacher's, and self-labels. Human labels have good quality but are scarce and costly to collect in large quantities. Teacher's labels are generated by a previously trained model and are referred to as distillation. Self-labels or proxy-labels are generated by the model itself. Label level refers to term-level or passage/document-level labels. Term-level labels provide one score per term, while passage-level labels indicate relevance for query-passage pairs. Most methods use passage-level labels.

2.2 Surveyed learned sparse retrieval methods

In Table 2, we present a summary of LSR methods fit into our conceptual framework. We cover nearly all transformer-based LSR methods for text ranking in the literature[4], but omit several due to time and space limitations [2, 4, 11, 25]. We group the methods into four groups by their conceptual similarity. We discuss how the methods fit into our framework and point out any small differences that are not described by our three components (e.g., choice of nonlinearity and including term or passage quality functions).

A. Methods without any expansion. DeepCT [5] and **uniCOIL** [16] use an MLP encoder for weighting terms in queries and documents, with a slight modification to Equation 2 by removing log normalization. Using the MLP means no expansion is applied (to query or document). DeepCT and uniCOIL only differ in supervision. DeepCT is supervised by term-recall, a term-level label defined as the ratio of relevant queries containing a term. On the other hand, uniCOIL uses passage-level labels rather than supervising individual term scores.

B. Methods without query expansion. uniCOIL$_{dT5q}$ [16], **uniCOIL$_{tilde}$** [16], and **EPIC** [20] replace the MLP document encoder in group **A** with either an $expMLP$ or MLM encoder, which is capable of document expansion. As a pre-processing step, uniCOIL$_{dT5q}$ and uniCOIL$_{tilde}$ expand passages with relevant terms generated by third-party models (docT5query [27], TILDE). Instead of pre-expanding the passages, EPIC is the first to leverage the MLM architecture trained to do document expansion and term scoring end-to-end at once. On the query side, EPIC keeps the log normalization as in Equation 2. On the document side, the ReLU in Equation 3 is replaced by a Softplus and both $q(t)$ and $g(t)$ are modeled by a linear layer with a softmax activation.

C. Methods without query expansion or weighting. DeepImpact [23], **Sparta** [41], **TILDE** [44], and **TILDEv2** [43] simplify methods in group **B** by removing the (MLP) query encoder, hence have a near-instant query encoding time but no query expansion and weighting capability. DeepImpact and TILDE$_{v2}$ can be viewed as the uniCOIL$_{dT5q}$ and uniCOIL$_{tilde}$ models without a query encoder, respectively. Sparta is simplified from EPIC by (1) removing query encoder and (2) removing $q(t)$ and $g(t_j)$ in Equation 3. TILDE replaces the MLM head in Sparta with clsMLM.

[4] We consider the prominent doc2query document expansion methods [27, 28] in the context of pre-processing for document expansion (e.g., combined with uniCOIL), but we do not treat these as standalone *retrieval* methods.

D. Methods with full expansion and weighting. Splade-max [7] and **distilSplade-max** [7] use a shared MLM architecture on both the query and document side. The MLM enables end-to-end weighting and expansion for both query and document. Instead of selecting top-k terms as in EPIC, this Splade family uses the FLOPs regularizer during training to sparsify the representations. The difference between Splade-max and distilSplade-max is the supervision. While Splade-max is trained with multiple in-batch BM25 negatives, distilSplade-max is trained with a distillation technique using mined hard negatives. Similar to Sparta, $q(t)$ and $g(t_j)$ in Equation 3 are removed from Splade models.

	Method	Query	Passage	Reg.	Supervision		
					Level	Neg.	Type
A	DeepCT [5]	MLP	MLP	-	Term	-	-
	uniCOIL [16]	MLP	MLP	-	Passage	BM25(s)	Human
B	uniCOIL$_{dT5q}$ [16]	MLP	expMLP	-	Passage	BM25(s)	Human
	uniCOIL$_{tilde}$ [16]	MLP	expMLP	-	Passage	BM25(s)	Human
	EPIC [20]	MLP	MLM	Top-k	Passage	BM25	Human
C	DeepImpact [23]	BINARY	expMLP	-	Passage	BM25	Human
	TILDE [43]	BINARY	clsMLM	-	Term	-	-
	TILDEv2 [43]	BINARY	expMLP	-	Passage	BM25(s)	Human
	Sparta [41]	BINARY	MLM	-	Passage	BM25	Human
D	SPLADE-max [7]	MLM	MLM	FLOPs	Passage	BM25(s)	Human
	DistilSPLADE-max [7]	MLM	MLM	FLOPs	Passage	Hard	Teacher

Table 2: Definition of existing LSR methods. An (s) indicates multiple negatives.

3 Experimental settings

For all experiments, we use Huggingface's BERT implementation with *distilbert-base-cased* [34, 37]. We train our models on the MSMarco [26] and TripClick datasets [31]. For models that need hard negative mining and distillation on MSMarco, we use the data provided by SentenceTransformers[5] [30] for training. For TripClick, we use the training triples[6] created by [9]. We evaluate methods with the benchmarks' standard metrics, including MRR@10, NDCG@10, and Recall@1000. In the following sections, we remove the cut-off @K for brevity.

We measure encoding latency on an AMD EPYC 7702 CPU and Tesla V100 GPU. We use a modified version of Anserini [15] for indexing passages and measure retrieval latency on an AMD EPYC 7702 CPU using 60 threads. For **RQ1**, we followed the same hyper-parameters and losses described in the original papers to reproduce LSR methods. For **RQ2** and **RQ3**, we train all methods on a single A100 GPU using the above mined hard negatives, and distillation data for MSMarco or the BM25 triplets for TripClick. Our Github repository contains the full configurations for all experiments.

[5] `huggingface.co/datasets/sentence-transformers/msmarco-hard-negatives`
[6] `github.com/sebastian-hofstaetter/tripclick`

	Method	Original MRR	Reproduced MRR	Δ %
A	DeepCT	24.3	24.6	1.234
	uniCOIL	31.5	31.6	0.317
B	uniCOIL$_{dT5q}$	35.2	34.7	-1.420
	uniCOIL$_{tilde}$	34.9	34.8	-0.286
	EPIC$^{*}_{top1000}$	27.3	28.8	5.495
C	DeepImpact	32.6	31.2	-4.294
	TILDE$^{*}_{v2}$	33.3	33.7	1.201
	Sparta	-	31.0	-
D	Splade$_{max}$	34.0	34.0	0.000
	distilSplade$_{max}$	36.9	37.9	2.439

Table 3: Reproduced MRR@10 scores on MSMarco dev. (*) Indicates reranking results on BM25 top-1000 passages (following the original work).

4 Results and analysis

In this section we consider our three RQs. We first reproduce LSR methods in their original experimental settings (RQ1), before training them in a common setting (RQ2) and analyzing the impact of architectural differences (RQ3).

4.1 RQ1: Are the results from LSR papers reproducible?

We train the LSR methods using a similar experimental setup described in the original papers and code. The reproduced results are reported in Table 3. For most of the methods, we obtain scores that are slightly higher or comparable to the original work. A slightly higher MRR was observed for DeepCT, uniCOIL, EPIC, TILDE$_{v2}$, and distilSplade$_{max}$, while DeepImpact and uniCOIL$_{dT5q}$ received slightly lower reproduced scores. Sparta was not evaluated on MSMarco in the original paper, so there is no comparison point for our result.

These reproduced results show that DeepCT and uniCOIL (without docT5query expansion) tend to be the least effective approaches, whereas distilSplade$_{max}$ achieves the highest MRR. Interestingly, we observe pairs of methods that have identical architectures, but different training recipes lead to a significant discrepancy in scores. uniCOIL changes the supervision signal of DeepCT from token-level weights to passage-level relevance, making a 28% jump in MRR from 24.6 to 31.6. Apparently, the supervision matters a lot here; using the passage-level labels allows the model to learn the term weights more optimally for passage-level relevance. Similarly, using mined hard negatives and distillation boosts MRR from 34.0 to 37.9 with the Splade model. This change of supervision makes distilSplade$_{max}$ the most effective LSR method considered. Without this advanced training, Splade$_{max}$ performs comparably to uniCOIL$_{dT5q}$ and uniCOIL$_{tilde}$. Looking closely at the group (**B**), EPIC seems to perform under its full capacity because it achieves a MRR substantially below the two uniCOIL variants. This may be due to the fact that EPIC was originally trained on 40000 triples, whereas the other methods were trained on up to millions of samples.

	Method	MSMarco		DL-2019		DL-2020		Index	RL	BM25 Negs	
		MRR	R	NDCG	R	NDCG	R	GB	ms	MRR	R
A	uniCOIL	$27.3^{\dagger\dagger\dagger}_{***}$	$88.0^{\dagger\dagger\dagger}_{***}$	59.3	72.9	54.3	77.9	**1.1**	**6.1**	32.1	92.6
B	uniCOIL$_{dT5q}$	$35.0^{\dagger}_{*}$	95.7	65.9	81.0	68.4	84.6	1.8	12.7	34.7	96.4
	uniCOIL$_{tilde}$	$36.1^{\dagger\dagger\dagger}_{***}$	$96.8^{\dagger\dagger}_{**}$	69.1	82.2	69.4	85.2	2.6	7.1	34.8	96.5
	EPIC$_{top400}$	$37.2^{\dagger\dagger\dagger}_{***}$	$97.2^{\dagger\dagger\dagger}_{***}$	70.9	87.7	71.8	88.7	9.7	17.7	**35.5**	96.4
C	DeepImpact	32.2_{**}	$94.7^{\dagger\dagger\dagger}$	63.1	77.2	63.3	82.1	1.8	16.1	32.2	95.4
	TILDE$_{top400}$	$29.9^{\dagger\dagger\dagger}$	$93.9^{\dagger\dagger\dagger}$	65.1	68.5	63.0	69.9	6.4	29.0	21.6	74.5
	TILDE$_{v2}$	$32.9^{\dagger\dagger}_{**}$	96.0	66.3	79.7	65.9	83.5	2.6	9.5	33.7	96.1
	Sparta$_{top400}$	$35.3^{\dagger\dagger\dagger}$	$96.8^{\dagger\dagger\dagger}$	69.1	81.9	68.1	85.8	6.1	26.7	28.3	88.7
D	distilSplade$_{max}$	37.9†††	98.1†††	**74.8**	**87.9**	**72.5**	**89.5**	6.3	122.5	35.3	**97.0**
	distilSplade$_{sep}$	**38.0**	98.0	74.1	87.7	70.6	89.0	8.0	50.2	-	-

****/† † † $p < 0.01$, **/†† $p < 0.05$, */† $p < 0.1$ with paired two-tailed t-test*
Comparing with results in Table 3 (*) and BM25 negatives results (†)

Table 4: Results with cross-encoder distillation on hard negatives (left) and BM25 negatives on MS MARCO (two rightmost columns). **RL** indicates the latency (ms/q) for query encoding and retrieval.

4.2 RQ2: How do LSR methods perform with recent advanced training techniques?

Variations in environments, as shown in RQ1, make it difficult to fairly compare LSR methods and can lead to inaccurate conclusions. To eliminate these discrepancies, we train all methods in a consistent environment, which we show to be effective in this section. We focus on the most effective supervision setup, which is distilSplade$_{max}$ trained using distillation and hard negatives. Table 4 shows the results of the LSR methods under this setting. Note that several methods (DeepCT and uniCOIL; Splade variants) will have identical scores in this experiment as they collapse into the same model. We only report a representative method in these cases.

Comparing to the results of **RQ1** (Table 3), we find that the least effective methods (DeepCT, now equivalent to uniCOIL) and the most effective method (distilSplade$_{max}$) remain in the same positions. Methods between these two endpoints move around with substantial changes in their effectiveness. Out of 10 methods we reproduced in Table 3, we observe an upward trend on seven methods, while the remaining three methods stay the same or perform worse. The biggest jumps are seen using EPIC and Sparta, with a relative improvement of 8.0 and 4.2 MRR points on MSMarco, respectively. The increase in EPIC's effectiveness, which is due to the combination of longer training time and improved supervision, moves the approach's relative ranking from the second worst to the second best, with metrics competitive with distilSplade$_{max}$ on MSMarco. On TREC DL 2019 and TREC DL 2020, the gap in NDCG@10 between EPIC and distilSplade$_{max}$ is higher. The increased MRR@10 on MSMarco also brings

Sparta a nice efficiency-effectiveness trade-off: since there is no query encoder with Sparta, there is no need for a GPU at retrieval time.

In addition to EPIC and Sparta, we also observe positive trends with DeepCT, DeepImpact, uniCOIL$_{dT5q}$ and uniCOIL$_{tilde}$; however, the change is relatively marginal. We observe decreased effectiveness on uniCOIL and TILDE$_{v2}$. While the decline with TILDE$_{v2}$ is small, the drop with uniCOIL (32.1→27.3) is quite large. Indeed, without expansion capability, no soft-matching could be possible, which renders a challenge for uniCOIL to reconstruct the MarginMSE's loss margin produced by a cross-encoder teacher, which is capable of soft-matching.

Regarding architecture types, methods using the MLM architecture, either on the document or query side (EPIC, Sparta, Splade), generally perform better than those using other architectures (clsMLM, MLP, expMLP, BINARY) on all three datasets. However, MLM also increases index size and latency significantly. For instance, EPIC's index is at least 6 GB larger than other methods in the group. Notably, distilSplade$_{max}$ not only creates a large index but also has a notably high retrieval latency, almost 20 times slower than the fastest method.

The latency issue in Splade is related to using the same shared MLM encoder for query and documents, resulting in similar term activation probability between queries and documents. We confirmed this by replacing the shared encoder with two separate ones (distilSplade$_{sep}$), which reduced latency from 122.5 ms to 50.2 ms, a 59% decrease. This benefit of separate encoders was also reported in [14], and our results further support its substantial impact.

In the last two columns of Table 4, we provide additional MSMarco results with training using BM25 in-batch negatives (the same as uniCOIL's original setup). We find that using hard negatives with distillation is generally more effective than using BM25 negatives, though not with uniCOIL or TILDE$_{v2}$.

4.3 RQ3: How does the choice of encoder architecture and regularization affect results?

In this RQ, we aim to quantify how different factors (*query expansion*, *document expansion*, *query weighting*, *document weighting*, *regularization*) affect the effectiveness and efficiency of LSR systems. To eliminate potential confounding factors due to minor differences between groups (e.g., choice of nonlinearity), we perform a series of controlled experiments in which we make single architectural changes while holding the rest of the architecture constant.

In Table 5, numbers before `+` or `-` are the metrics before a change (left side of arrow), while numbers after these symbols show the effect of a change (right). We see that document weighting seems to be the most crucial component since the systems without this component fail on all three datasets. In row $1_{(a,b)}$, the system with a binary document encoder shows very low MRR and NDCG scores regardless of MLM or MLP on the query side. On both MSMarco and TripClick, enabling document weighting (by replacing the binary document encoder with an MLP) improves the effectiveness by a large margin (at least 11 points) with reasonable latency and index size increases. Without document weighting, the models are not able to identify important terms in documents.

Effect		Control	Change	MSMarco MRR	MSMarco R	DL 2019 NDCG	DL 2020 NDCG	Latency ms	Index GB
Doc weighting	1_a	Q_{MLM}	$D_{BIN} \rightarrow D_{MLP}$	16.7+**18.3**	86.0+**11.0**	44.1+**26.8**	42.9+**24.5**	11.4+**04.2**	0.6+**0.7**
	1_b	Q_{MLP}	$D_{eBIN} \rightarrow D_{eMLP}$	08.2+**27.9**	76.2+**20.6**	30.4+**38.7**	27.5+**41.9**	10.8−**03.7**	1.2+**1.4**
Query weighting	2_a	D_{eMLP}	$Q_{BIN} \rightarrow Q_{MLP}$	32.9+**3.2**	96.0+**0.8**	66.3+**2.8**	65.9+**3.5**	09.5−**0.9**	2.6+**0.0**
	2_b	D_{MLM}	$Q_{BIN} \rightarrow Q_{MLP}$	35.2+**1.9**	96.5+**0.7**	69.4+**1.5**	69.7+**2.1**	28.9−**7.9**	8.6+**1.1**
Doc expansion	3_a	Q_{MLM}	$D_{MLP} \rightarrow D_{MLM}$	34.9+**3.1**	97.0+**0.9**	70.9+**3.3**	67.4+**3.2**	15.6+**34.6**	1.3+**6.7**
	3_b	Q_{MLP}	$D_{MLP} \rightarrow D_{MLM}$	27.5+**10.0**	89.7+**8.2**	59.3+**12.0**	54.3+**17.9**	27.5+**10.5**	1.2+**6.9**
Query expansion	4_a	D_{MLM}	$Q_{MLP} \rightarrow Q_{MLM}$	38.0+**0.0**	97.0+**0.1**	71.3+**2.8**	72.1−**1.3**	12.9+**37.3**	8.0−**0.1**
	4_b	D_{MLP}	$Q_{MLP} \rightarrow Q_{MLM}$	27.5+**7.5**	89.7+**7.4**	59.3+**11.6**	54.3+**13.1**	06.1+**9.5**	1.2+**0.1**
Regularization	5_a	Q_{MLP} D_{MLM}	$FLOPs \rightarrow Topk$	38.0+**0.0**	97.9−**0.3**	71.3+**0.8**	72.1+**0.1**	12.8+**4.3**	8.1−**0.7**

				TripClick HEAD(dctr) NDCG	TripClick HEAD(dctr) R	TripClick TORSO(raw) NDCG	TripClick TAIL(raw) NDCG	Latency ms	Index GB
Doc weighting	1_a	Q_{MLP}	$D_{BIN} \rightarrow D_{MLP}$	6.5+**18.9**	69.7+**18.4**	10.7+**17.5**	16.2+**13.2**	2.0−**0.1**	0.3+**0.3**
	1_b	Q_{MLP}	$D_{eBIN} \rightarrow D_{eMLP}$	5.7+**21.0**	67.2+**21.1**	9.1+**20.4**	13.9+**16.5**	2.5−**0.3**	0.4+**0.5**
Query weighting	2_a	D_{MLM}	$Q_{BIN} \rightarrow Q_{MLP}$	26.3+**3.9**	90.0+**1.9**	31.3+**3.3**	34.2+**3.8**	3.2−**0.0**	1.8−**0.1**
	2_b	D_{MLP}	$Q_{BIN} \rightarrow Q_{MLP}$	24.2+**1.1**	87.3+**0.8**	27.7+**0.4**	29.4+**0.0**	2.1−**0.2**	0.5+**0.1**
Doc expansion	3_a	Q_{MLM}	$D_{MLP} \rightarrow D_{MLM}$	27.9+**2.2**	90.9+**1.0**	32.7+**1.5**	34.1+**3.9**	4.6+**1.6**	0.7+**0.7**
	3_b	Q_{MLP}	$D_{MLP} \rightarrow D_{MLM}$	25.3+**4.7**	88.1+**3.7**	28.2+**6.1**	29.4+**7.9**	1.9+**1.6**	0.6+**0.8**
Query expansion	4_a	D_{MLM}	$Q_{MLP} \rightarrow Q_{MLM}$	30.0+**0.1**	91.8+**0.1**	34.2−**0.1**	37.4+**0.6**	3.4+**2.8**	1.4−**0.0**
	4_b	D_{MLP}	$Q_{MLP} \rightarrow Q_{MLM}$	25.3+**2.6**	88.1+**2.8**	28.2+**4.5**	29.4+**4.6**	1.9+**2.7**	0.6+**0.0**
Regularization	5_a	Q_{MLP} D_{MLM}	$L1 \rightarrow Topk$	30.0+**0.1**	91.8+**0.1**	34.2+**0.3**	37.4+**0.7**	3.4−**0.2**	1.4+**0.3**

Table 5: The effects of architecture and regularizer on MSMarco and TripClick. *We use names that better reflect the architectural differences between methods. Visit our Github repository to see the full configurations and original names.*

Similarly, as shown in rows $2_{(a,b)}$, we control the document side and change the binary query encoder to an MLP query encoder to observe the effect of query weighting. The result suggests that query weighting has a moderate contribution to the ranking metrics overall. Still, interestingly, it causes almost no harm to the index size or even reduces the latency. Note that the latency of the MLP query encoder here is measured on GPU; therefore, the encoding overhead is tiny. The improved overall latency is mostly due to the MLP reducing the weights of some non-useful query terms to zero, making queries shorter. The effect is quite consistent between MSMarco and TripClick collections.

Regarding the expansion factors, we observe the cancellation effect between query expansion and document expansion. Indeed, with the absence of expansion on one side (3_b: Q_{MLP} has no query expansion, 4_b: D_{MLP} has no document expansion), the expansion on the other side largely improves the ranking metrics with at least 7.4(2.6) points and at most 17.9(7.9) points overall on MSMarco (TripClick). The cost of latency, in this case, is rather low. The numbers in rows 3_a and 4_a indicate that query and document expansion have a cancellation effect. That is, query expansion reduces the benefit of performing document expansion and vice versa. Row 4_a shows that when document expansion is in place, query expansion has minimal impact on ranking effectiveness and incurs a relatively high latency overhead (increases of 289% and 82% on MSMarco and TripClick). Row 3_a shows a similar trend, with document expansion making

moderate contributions to system effectiveness. On TripClick, the cancellation interaction between the two factors is less strong. Overall, this cancellation effect suggests that including both expansion components may not be necessary.

Lastly, to examine the effect of regularization, we keep the model's architecture constant and change the FLOPs/L_1 regularizer during training to Topk pruning during inference. As shown in rows 5_a, changing the regularization approach does not significantly affect effectiveness or efficiency.

Method	MSMarco-dev				TripClick-HEAD(dctr)			
	MRR	R	Index(GB)	RL(ms)	NDCG	R	Index(GB)	RL(ms)
distilSplade$_{sep}$	38.0	98.0	8.0	50.2	30.1	91.9	1.4	6.3
distilSplade$_{qMLP}$	38.0	97.9	8.1	12.9	30.0	91.8	1.4	3.4
distilSplade$_{dMLP}$	34.9*	97.0*	1.3	15.6	27.9*	90.9*	0.7	4.6

Table 6: Results with only query expansion or only document expansion.
* $p < 0.01$ *with paired two-tailed t-test*

In Table 6, we show the results of systems with expansion only on either the query or the document side. In the table, distilSplade$_{qMLP}$ denotes the distilSplade$_{sep}$ with the MLM query encoder replaced by an MLP query encoder; hence no query expansion is involved. Similar interpretation applies for distilSplade$_{dMLP}$. As can be seen, distilSplade$_{qMLP}$ makes no significant changes on ranking metrics, while reducing the retrieval latency by more than 74% and 46% on MSMarco and TripClick, respectively. distilSplade$_{dMLP}$ exhibits a similar latency improvement, but suffers from a significant drop in effectiveness. In practice, distilSplade$_{qMLP}$ could be viewed as a more efficient drop-in replacement for the full model. This use of $qMLP$ is complementary to other changes (e.g., using a smaller encoder as in [14]) to improve the efficiency of LSR.

5 Conclusion

In this work, we introduced a conceptual framework for learned sparse retrieval that unifies existing LSR methods under the perspective of three components. After reproducing these methods, we carried out a series of experiments to isolate the effect of single changes on a model's performance. This analysis led to several findings about the components, including that we can remove the query expansion from a SOTA system, leading to a significant latency improvement without compromising the system's effectiveness. While this study covered the most prominent transformer-based LSR methods, several others could not be considered due to time and computing constraints (e.g., [2, 4, 11, 25]). We plan to incorporate them into our implementation as future work.

Acknowledgement

We thank Maurits Bleeker from the UvA IRLab for his feedback on the paper.

References

1. Ash, J.T., Goel, S., Krishnamurthy, A., Misra, D.: Investigating the role of negatives in contrastive representation learning. arXiv preprint arXiv:2106.09943 (2021)
2. Bai, Y., Li, X., Wang, G., Zhang, C., Shang, L., Xu, J., Wang, Z., Wang, F., Liu, Q.: Sparterm: Learning term-based sparse representation for fast text retrieval. arXiv preprint arXiv:2010.00768 (2020)
3. Chen, X., Lakhotia, K., Oğuz, B., Gupta, A., Lewis, P., Peshterliev, S., Mehdad, Y., Gupta, S., Yih, W.t.: Salient phrase aware dense retrieval: Can a dense retriever imitate a sparse one? arXiv preprint arXiv:2110.06918 (2021)
4. Choi, E., Lee, S., Choi, M., Ko, H., Song, Y.I., Lee, J.: Spade: Improving sparse representations using a dual document encoder for first-stage retrieval. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management. pp. 272–282 (2022)
5. Dai, Z., Callan, J.: 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. pp. 1533–1536 (2020)
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). pp. 4171–4186 (2019)
7. Formal, T., Lassance, C., Piwowarski, B., Clinchant, S.: Splade v2: Sparse lexical and expansion model for information retrieval. arXiv preprint arXiv:2109.10086 (2021)
8. Formal, T., Lassance, C., Piwowarski, B., Clinchant, S.: From distillation to hard negative sampling: Making sparse neural ir models more effective. arXiv preprint arXiv:2205.04733 (2022)
9. Hofstätter, S., Althammer, S., Sertkan, M., Hanbury, A.: Establishing strong baselines for tripclick health retrieval. In: European Conference on Information Retrieval. pp. 144–152. Springer (2022)
10. Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., Grave, E.: Unsupervised dense information retrieval with contrastive learning. arXiv preprint arXiv:2112.09118 (2021)
11. Jang, K.R., Kang, J., Hong, G., Myaeng, S.H., Park, J., Yoon, T., Seo, H.: Ultra-high dimensional sparse representations with binarization for efficient text retrieval. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 1016–1029 (2021)
12. Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., Yih, W.t.: Dense passage retrieval for open-domain question answering. arXiv preprint arXiv:2004.04906 (2020)
13. Khattab, O., Zaharia, M.: 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. pp. 39–48 (2020)
14. Lassance, C., Clinchant, S.: An efficiency study for splade models. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2220–2226 (2022)
15. Lin, J., Crane, M., Trotman, A., Callan, J., Chattopadhyaya, I., Foley, J., Ingersoll, G., Macdonald, C., Vigna, S.: Toward reproducible baselines: The open-source ir

reproducibility challenge. In: European Conference on Information Retrieval. pp. 408–420. Springer (2016)
16. Lin, J., Ma, X.: A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques. arXiv preprint arXiv:2106.14807 (2021)
17. Lin, J., Nogueira, R., Yates, A.: Pretrained transformers for text ranking: Bert and beyond. Synthesis Lectures on Human Language Technologies **14**(4), 1–325 (2021)
18. Lin, S.C., Lin, J.: Densifying sparse representations for passage retrieval by representational slicing. arXiv preprint arXiv:2112.04666 (2021)
19. Lin, S.C., Lin, J.: A dense representation framework for lexical and semantic matching. arXiv preprint arXiv:2206.09912 (2022)
20. MacAvaney, S., Nardini, F.M., Perego, R., Tonellotto, N., Goharian, N., Frieder, O.: Expansion via prediction of importance with contextualization. In: Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval. pp. 1573–1576 (2020)
21. Mackenzie, J., Trotman, A., Lin, J.: Wacky weights in learned sparse representations and the revenge of score-at-a-time query evaluation. arXiv preprint arXiv:2110.11540 (2021)
22. Mackenzie, J., Trotman, A., Lin, J.: Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. ACM Trans. Inf. Syst. (dec 2022). https://doi.org/10.1145/3576922, `https://doi.org/10.1145/3576922`, just Accepted
23. Mallia, A., Khattab, O., Suel, T., Tonellotto, N.: Learning passage impacts for inverted indexes. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1723–1727 (2021)
24. Mallia, A., Mackenzie, J., Suel, T., Tonellotto, N.: Faster learned sparse retrieval with guided traversal. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. p. 1901–1905. SIGIR '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3477495.3531774, `https://doi.org/10.1145/3477495.3531774`
25. Nair, S., Yang, E., Lawrie, D., Mayfield, J., Oard, D.W.: Learning a sparse representation model for neural clir (2022)
26. Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., Deng, L.: Ms marco: A human generated machine reading comprehension dataset. In: CoCo@ NIPs (2016)
27. Nogueira, R., Lin, J.: From doc2query to docTTTTTquery (2019)
28. Nogueira, R., Yang, W., Lin, J., Cho, K.: Document expansion by query prediction. arXiv preprint arXiv:1904.08375 (2019)
29. Paria, B., Yeh, C.K., Yen, I.E., Xu, N., Ravikumar, P., Póczos, B.: Minimizing flops to learn efficient sparse representations. In: International Conference on Learning Representations (2019)
30. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics (11 2019), `https://arxiv.org/abs/1908.10084`
31. Rekabsaz, N., Lesota, O., Schedl, M., Brassey, J., Eickhoff, C.: Tripclick: the log files of a large health web search engine. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2507–2513 (2021)

32. Robertson, S., Zaragoza, H., et al.: The probabilistic relevance framework: Bm25 and beyond. Foundations and Trends® in Information Retrieval **3**(4), 333–389 (2009)
33. Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M., Gatford, M.: Okapi at TREC-3. In: Harman, D.K. (ed.) Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994. NIST Special Publication, vol. 500-225, pp. 109–126. National Institute of Standards and Technology (NIST) (1994), `http://trec.nist.gov/pubs/trec3/papers/city.ps.gz`
34. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108 (2019)
35. Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., Gurevych, I.: Beir: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In: Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2) (2021)
36. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
37. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al.: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations. pp. 38–45 (2020)
38. Xiong, L., Xiong, C., Li, Y., Tang, K.F., Liu, J., Bennett, P., Ahmed, J., Overwijk, A.: Approximate nearest neighbor negative contrastive learning for dense text retrieval. arXiv preprint arXiv:2007.00808 (2020)
39. Yang, P., Fang, H., Lin, J.: Anserini: Reproducible ranking baselines using lucene. Journal of Data and Information Quality (JDIQ) **10**(4), 1–20 (2018)
40. Zamani, H., Dehghani, M., Croft, W.B., Learned-Miller, E., Kamps, J.: From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In: Proceedings of the 27th ACM international conference on information and knowledge management. pp. 497–506 (2018)
41. Zhao, T., Lu, X., Lee, K.: Sparta: Efficient open-domain question answering via sparse transformer matching retrieval. arXiv preprint arXiv:2009.13013 (2020)
42. Zheng, G., Callan, J.: Learning to reweight terms with distributed representations. In: Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval. pp. 575–584 (2015)
43. Zhuang, S., Zuccon, G.: Fast passage re-ranking with contextualized exact term matching and efficient passage expansion. arXiv preprint arXiv:2108.08513 (2021)
44. Zhuang, S., Zuccon, G.: Tilde: Term independent likelihood model for passage re-ranking. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1483–1492 (2021)