Adaptive Sparsity Optimization with Learnable Soft Top-K and Per-Term Thresholding for Efficient Retrieval

Wentai Xie
University of California, Santa Barbara
Santa Barbara, California, USA
wentaixie@ucsb.edu

Parker Carlson
University of California, Santa Barbara
Santa Barbara, California, USA
parker_carlson@ucsb.edu

Shanxiu He
University of California, Santa Barbara
Santa Barbara, California, USA
shanxiuhe@ucsb.edu

Tao Yang
University of California, Santa Barbara
Santa Barbara, California, USA
tyang@cs.ucsb.edu

Abstract

Recent work on neural sparse retrieval has demonstrated strong relevance by leveraging Large Language Models (LLMs) for semantic term expansion. However, learned models paired with previous sparsification techniques still yield overly long document and query vectors partly due to a large LLM vocabulary, imposing a serious challenge to retrieval time and space efficiency. This paper proposes a scheme for optimizing model sparsity through a synergy of adaptive strategies, including learnable soft top-K, per-term thresholding, and FLOPs regularization to increase the sparsity of query and document vectors. Experimental results with Lion-SP model on the MS MARCO and BEIR datasets demonstrate that the proposed scheme can outperform the baselines by significantly reducing the average query and document lengths. Our scheme can achieve much shorter retrieval latency and lower storage cost while maintaining highly competitive relevance.

CCS Concepts

• **Information systems** → **Learning to rank**.

Keywords

Learned Sparse Representations; Sparse Retrieval; Sparsification; Efficiency

ACM Reference Format:
Wentai Xie, Parker Carlson, Shanxiu He, and Tao Yang. 2026. Adaptive Sparsity Optimization with Learnable Soft Top-K and Per-Term Thresholding for Efficient Retrieval . In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '26), July 20–24, 2026, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3805712.3809625

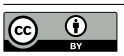
1 Introduction

Large-scale text search and retrieval-augmented generation [24] depend on efficient document retrieval to identify the top-k relevant results for re-ranking or LLM-based inference. Learned sparse retrieval techniques [11, 13–16, 26, 29, 38] based on BERT have demonstrated strong effectiveness in neural retrieval. These methods leverage transformer-based models to expand document tokens and assign learned weights, producing sparse representations with high relevance. At the same time, they can take advantage of traditional inverted index based retrieval techniques [29, 30] , enabling efficient retrieval on low-cost CPU platforms.

Recently, the Lion-SP model [39] represents a shift toward using large-scale decoder-only architectures for sparse vector generation with LLaMA-3 [17]. Although Lion adopts the SPLADE methodology by using FLOPs regularization [15, 20, 31] to minimize non-zero weights, Lion-SP still generates vectors with an outsized number of non-zero tokens (8.9x longer document vectors compare to one of the SPLADE variance [33]). This is partly due to LLaMA-3's 128,256-token vocabulary, which is 4.2× larger than BERT's. Although Lion-SP achieves state-of-the-art performance on BEIR and MS MARCO, it remains challenging to substantially improve its sparsity while preserving the relevance advantage provided by LLaMA-3. Crucially, sparsity directly affects retrieval efficiency. In sparse retrieval, query processing latency and inverted-index storage generally scale with the number of non-zero terms in the query and document representations.

Previous research on static inverted index pruning [4, 7, 9, 23] has studied term-centric and document-centric strategies to trim a term posting list or a document vector by a hard limit. Top token selection [37, 40] limits the highest activated weight counts uniformly across all documents. Mass ratio pruning [25] is a contemporary effort to preserve top entries using the vector mass [6]. Hybrid thresholding (HT) [33] exploits a learnable term-centric thresholding architecture to filter out unimportant neural weights. These existing techniques offer limited learnability and insufficient fine-grained context-aware adaptivity, which limits their ability to produce substantially sparser LLM-driven representations.

To address this challenge, this paper proposes AdaSparse, short for "Adaptive Sparsification". This scheme achieves fine-grained pruning optimization through a synergy of adaptive strategies, including learnable soft per-vector top-K and per-term thresholding, which work in complement with FLOPs regularization. A key distinction of this soft top-K technique is that it avoids explicit, hard pruning after the K-th ranked term of a query or document vector. This mechanism allows the model to capture sparsity penalty signals and autonomously adjust term weights during refinement.



SIGIR '26, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-2599-9/2026/07
https://doi.org/10.1145/3805712.3809625

AdaSparse is evaluated with Lion-SP for the MS MARCO passage dataset in training and testing, and for BEIR datasets in zero-shot retrieval. The experiments show that AdaSparse can generate much shorter sparse query and document vectors by a 4x-5x reduction in expanded query and document lengths while retaining highly competitive relevance. It can reduce actual retrieval latency by 3.8x and storage space by 4.2x using a state-of-the-art Seismic retriever [6].

2 Background and Related Work

Learned sparse retrieval. Given a query, document retrieval finds top-k ranked results by computing the similarity between a query vector and a document vector. These query vectors are generated during online inference, while document vectors are pre-generated during offline index processing based on a learned neural transformer-based model.

We use "tokens" and "terms" interchangeably to describe the units produced by a language model's tokenizer. The SPLADE family [13–15] expands given query and document vectors to address vocabulary mismatch due to the lack of semantically-related terms in the original text. SPLADE predicts term importance by mapping an input query or document sequence S into a high-dimensional sparse vector $\mathbb{R}^{|V|}$, where V is the size of the vocabulary. These models project the transformer-based contextualized representations multiplied by the embedding table of the transformer to obtain intermediate weight $w_{i,j}$, where $w_{i,j}$ denotes the importance of i-th input term in S for each j-th term t_j in V. To produce the final sparse vector, log-smoothing and a ReLU activation are applied to ensure non-negativity, followed by a max-pooling operation across all input positions:

$$w_j = \max_{i \in S} \log(1 + \text{ReLU}(w_{i,j})) \tag{1}$$

The final rank score for each document during retrieval is a dot product between query and document vector representations. For sparse vectors with many zeros, online retrieval can utilize a data structure called inverted index for fast low-cost inference [29, 30].

The loss function of SPLADE models [13–15] contains a ranking loss L_R and FLOPs-based sparsity regularization [31]. The ranking loss has evolved from a log likelihood-based function for maximizing positive document probability to Margin-MSE [18] and/or KL Divergence for knowledge distillation. Lion [39] adapts the LLaMA architecture with bi-directional attention following LLM2Vec [3] and trains dense or sparse retrieval models with knowledge distillation. This paper focuses on its sparse model called Lion-SP, which adopts a formula similar to SPLADE with the FLOPs regularization in the loss function for representation sparsification.

The rank loss with FLOPs regularization used in Lion-SP and SPLADE is summarized as follows. Let B be a set of training queries with N documents involved in a batch. The overall loss for this batch including L_R, the rank loss, with knowledge distillation is:

$$L = (\frac{1}{|B|} \sum_{q \in B} L_R) + \lambda_Q L_Q + \lambda_D L_D \tag{2}$$

where L_Q and L_D are FLOPs for queries and documents:

$$L_Q = \sum_{t_j \in V} (\frac{1}{|B|} \sum_{q \in B} w_j^q)^2; \;\; L_D = \sum_{t_j \in V} (\frac{1}{N} \sum_{d=1}^{N} w_j^d)^2. \tag{3}$$

Related work. Other sparsification studies [6, 20, 25, 33, 37, 40] will be discussed in next section. In addition to SPLADE, there are other learned sparse retrieval models [11, 16, 26, 29, 35, 38] mainly based on BERT. Sparse autoencoder studies interpret neural networks by decomposing hidden representations into disentangled sparse features [5, 19] with a fixed sparsity constraint using top K selection when optimizing representation reconstruction [2, 19, 32]. Sparsification in VDR [40] for text-to-text and cross-model retrieval uses a hard top-K limit while considering the inclusion of original words. CSurF [12] studies contextual sparse representation, which accounts for the existence of original words.

3 Design Considerations

Sparsification for Lion-SP faces a distinct challenge due to the massive LLaMA-3 128K vocabulary, which is 4.2× larger than that of BERT-based SPLADE. This larger token space makes it harder to distinguish and suppress unimportant terms. More importantly, the vocabulary composition of LLaMA-3 further increases representation redundancy. First, the LLaMA-3 vocabulary assigns separate tokens to different variants of the same English word [34]. Its tokenizer tends to produce more variants than BERT, such as case variations. As a result, when a word is considered relevant, multiple variants of that word may also be marked as relevant. This makes the resulting query or document vector unnecessarily dense. Second, LLaMA-3 is pre-trained for multilingual conversation, whose objective is not fully aligned with our retrieval setting. As a result, the multilingual knowledge acquired during pre-training may cause the model to activate semantically related non-English tokens even in an English-only setting, unnecessarily increasing the length of the generated sparse vectors.

The above characteristics of LLaMA-3 likely cause the sparse vectors generated by Lion-SP to contain many low-weight, semantically redundant term scores. As illustrated in Figure 1, a large fraction of the weights generated by Lion-SP on MS MARCO passages are small and similar in value. Table 1 compares a randomly sampled sparse query vector generated by SPLADE, Lion-SP-1B, and our work AdaSparse applied to Lion-SP-1B. Columns 3 and Column 4 are the number of variants generated by each model for keywords "years" and "governor" in this query. SPLADE with BERT generates a vector of length 20, containing two variants of "governor" (singular and plural). Lion-SP-1B generates a vector of length 272 containing 9 English variants of the query word "year" (singular, plurals, uppercase, lowercase, having space before the term or not) and two Chinese terms corresponding to "year". Compare to SPLADE, it has a wider weight value range and a lower average value. AdaSparse produces a condensed version with 4 English variants of "years" and 3 variants of "governor", while also shrinking the term weight range and yielding an average term score closer to that of SPLADE.

To bridge the gap between the strong relevance of sparse representation generated by Lion-SP and the efficiency constraints of online systems, we consider several representative sparsification strategies that can reduce representation size after training. Table 2 compares the pros and cons of the four main previously proposed sparsification methods in its first four rows, which is elaborated below:

Table 1: Redundancy and weight range of a sparse vector generated for query "how many years did william bradford serve as governor of plymouth colony?"

Model	Length	#years	#governor	Min-Max	Avg
SPLADE	20	1	2	0.03 - 1.15	0.40
Lion-SP-1B	272	11	8	0.01 - 1.28	0.20
AdaSparse-1B	64	4	3	0.07 - 1.27	0.41

- **Fixed top term selection** [37, 40] retains a constant number or percentage of terms for each input query or document vector. While computationally efficient, these two top term selection methods do not account for context-aware weight distribution and their impact on relevance. For example, it may over-prune a complex, multi-faceted technical query (losing vital expansion terms) while under-prune a simple navigational query containing noisy and irrelevant terms. The α-mass cutoff [6] is proposed to retain the smallest subset of top entries whose cumulative sum reaches fraction α of the L_1 norm of the summary vectors of indexed clusters. This concept is extended recently to Mass Ratio Pruning (MRP) [25] for more adaptive top term selection of document and query vectors. This mass ratio is a fixed hyperparameter, and the pruning scope for each vector is not learnable.
- **FLOPs** [31], as used in Lion-SP [39] , regularizes based on average term weights in a training batch. While effective for overall macro-level efficiency by shrinking large noisy weights, it does not consider the relative semantic importance of terms at a fine-grained level. Another limitation is that the FLOPs regularizer can produce very small gradients for terms with already small weights, making it difficult to further suppress these weights to zero. Thus, FLOPs tends to leave a long tail of very small weights unless FLOPs regularization is applied heavily, which in turn affects relevance. An efficient version of SPLADE [20] studies several optimization tradeoffs including L_1 regularization for query vectors while keeping FLOPs for documents.
- **Learnable uniform term thresholding (HT [33])** uses two learned thresholds for queries and for documents respectively, and prunes a term when its weight is below such a global threshold. This addresses the weakness of FLOPs in handling very small term weights. However, the distributions of neural weights vary substantially across terms, using a global threshold is too coarse-grained to handle such diversity effectively. This limitation is further amplified in zero-shot retrieval, where applying a global threshold tuned on one domain to another domain may suppress all terms in some vectors, resulting in empty representations.

With the above in mind, we identify three design goals:

- **Learnability for better adaptiveness**. Sparsification needs to incorporate context-aware optimization that adapts to distribution variance in different query and document vectors for effective in-domain and out-of-domain search. We seek a learnable top-K method with a soft limit, tailored towards each query or document vector.
- **Fine-grained optimization**. Since many terms have small and similar values, the proposed scheme also needs to address this problem in a pool of 128K distinct tokens. Given the weakness of uniform thresholding, we propose to extend the HT method by introducing individualized thresholds for each term, which enables more fine-grained adaptation to term-specific weight distributions.
- **Multi-objective sparsification regularization**. As discussed later and shown in the last two rows of Table 2, we find that the two proposed new methods and the FLOPs method are complementary. This suggests the need to design a multi-objective loss function through a synergy of multiple methods.

4 The Proposed Methods for AdaSparse

Motivated by the design goals in Section 3, AdaSparse is built on two complementary sparsification techniques. To introduce adaptiveness into vector-level pruning, we propose STop, a learnable soft top-K method that provides a context-dependent soft limit for each query or document vector. To improve fine-grained pruning of low-weight terms, we introduce PTT, a per-term thresholding scheme that suppresses low-weight terms using individualized thresholds. These two techniques are complementary to FLOPs regularization: STop mainly controls the expansion scale, while PTT refines the pruning of weak terms included during generation. This section presents the two sparsification techniques and discusses how to integrate them with FLOPs in sparsity regularization.

4.1 STop: Learnable soft top-K

STop allows the model to adaptively select top-K important terms based on the context. Unlike fixed top term selection [6, 25, 37, 40], STop does not explicitly prune terms during training. Instead, it introduces a differentiable top-K selection signal that penalizes terms beyond a context-dependent soft limit, while allowing the model to adjust the effective number of retained terms for each vector. We now describe how this soft limit is defined, how the gating function is constructed, and used.

Step 1. Select a soft limit K. First, we set a soft limit for top-token selection for each vector:

$$K = b \cdot |s|$$

where s is the input vector and $|s|$ denotes its length. b is a hyper-parameter that determines the soft expansion factor. This ensures that the expansion of a document or a query is reasonably rich to provide a good bridge in query-document semantic matching. In practice, the actual expansion ratio may be smaller or slightly larger than the b value used, as shown in Section 5.

Step 2. Define a gating function for a document or query vector. Using the soft limit defined above, we introduce a sigmoid-based gating function G_j that compares each term weight with the weight around the K-th ranked position. This function produces a differentiable soft mask, which penalizes terms expanded beyond the soft top-K boundary during training:

$$G(w_j, v) = \sigma\left(\frac{\widehat{w}_K - w_j - \beta \cdot \mathbb{I}(t_j \in v_{\text{orig}})}{\gamma}\right) \tag{4}$$

where

- The sigmoid function ranges between 0 and 1: $\sigma(x) = \frac{1}{1+e^{-x}}$.
- w_j: The raw importance score for term t_j generated by the LLM expansion head.

Table 2: Comparison of Query Pruning Methodologies

Method	Mechanism	Strength	Weakness
FLOPs	Lower squared avg. term weight sum	Quickly shrink large noisy weights	Leave long tails of small weights
Top-K	Vector-centric, fixed K or K%	Constant sparsity	Not learnable K. Over-prune complex vectors
Mass ratio pruning	Vect.-centric. Top terms with mass ratio	Mass-adaptive sparsity	Not learnable mass ratio
Uniform thresholding	Term-centric, $w_j > \tau$	Relative importance filtering	Nonadapt. to distribut. diversity. Empty vect. risk
Per-term thresholding	Term-centric, $w_j > \tau_j$	Individualized, value-adaptive	Difficult for flat distribut.&reaching high sparsity
Learnable soft top-K	Vector-centric, penalty-driven soft limit	Adaptive without explicit pruning	Might be impacted by noisy term rankings

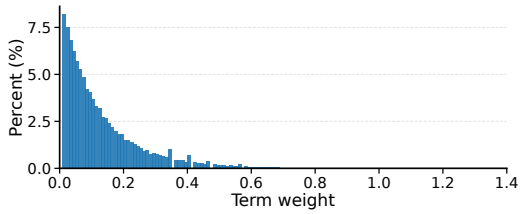


Figure 1: Term score distribution of Lion-SP 1B

- $\widehat{w}_K$: The **rank-relative pivot** which is the average term weight at the K-th and $K+1$-th ranked positions. Namely $\widehat{w}_K = (w_K + w_{K+1})/2$. It acts as a floating limit that adapts to the vector's specific semantic density.
- γ: A temperature parameter that controls the steepness of the gating boundary.
- β: A positive scalar called the **lexical provenance bias** that ensures terms found in the original, unexpanded vector v_{orig} have a significantly higher probability that the gating function returns a 0 or small value.
- $\mathbb{I}(t_j \in v_{\textbf{orig}})$: An indicator function that returns 1 if term t_j was part of the original vector and 0 otherwise.

We choose hyperparameter β to be bigger than the pivot value of all query and document vectors, namely $\beta > max(\widehat{w}_K)$, so that original terms are effectively exempted from the gating penalty. Even at the beginning of the training, $\widehat{w}_K$ may exceed β, $\widehat{w}_K$ value decreases gradually during iterative weight updating and the lexical bias can still make an impact. Based on the term-weight distribution shown in figure 1, where the nonzero scores of Lion-SP on MS MARCO are typically below 1.4, we set $\beta = 2$. Our training uses a scaling factor $\gamma = 0.04$ to accelerate the steepness of the sigma function curve. Under these parameter choices, the gating function has the following properties.

- We consider a term that t_j appears in the top K of the ranked term list of vector v; or when a term t_j appears in the original vector without expansion($\beta > \widehat{w}_K$). The former condition gives $\widehat{w}_K < w_j$ while the latter gives $\widehat{w}_K < \beta$. Thus, the input of the gating function $\frac{\widehat{w}_K - w_j - \beta \cdot \mathbb{I}(t_j \in v_{\text{orig}})}{\gamma} < 0$ and the output value $G(w_j, v) \approx 0$.
- When $\widehat{w}_K \approx w_j$ and term t_j is not in the original text before expansion, then $G(w_j, v) \approx \sigma(0) \approx 0.5$. This represents uncertainty of importance for terms without a clear decrease in value after the K-th position.
- Otherwise, $\frac{\widehat{w}_K - w_j - \beta \cdot \mathbb{I}(t_j \in v_{\text{orig}})}{\gamma} > 0$, and the gating function approaches 1 rapidly with very small γ factor. $G(w_j, v) \approx 1$.

Step 3. Apply gating function $G(w_j, v)$. We only use it for model training in STop loss regularization L_{top} defined as follows:

$$L_{top} = \sum_{t_j \in V} \left(\frac{\lambda_{qtop}}{|B|} \sum_{q \in B} (w_j^q)^2 G(w_j, q) + \frac{\lambda_{dtop}}{N} \sum_{d=1}^{N} (w_j^d)^2 G(w_j, d) \right) \tag{5}$$

where B is a set of training queries with N documents involved in a batch. λ_{qtop} and λ_{dtop} are coefficient hyperparameters. The above formula is essentially a gated L_2 loss, which will be added to the overall regularization loss Formula (8) to be discussed shortly.

The above design offers three distinct advantages:

- **Loss-driven term weight refinement without explicit top-K pruning**. Gating function G is not used to explicitly prune anything in training. It sends a strong signal through L_{top} based on the three possible scenarios of the function value discussed above. Thus, the soft top-K regularizer acts like a selective gravity. It is "turned off" for the top-K weights because $G(w_j, v) \approx 0$ for these weights. For other weights w_j (soft position $K+1$ and below) with $G(w_j, v) \approx 1$, which may trigger a decrease in w_j value during gradient-based weight updates.
- **Differentiability with value-based and count-adaptable soft limit**. The above gating function is differentiable and the rank-relative pivot is value-based, which allows a soft limit with count-adaptivity, rather than a hard limit. As shown later in the gradient analysis of Section 4.3, that can lead to a negative gradient for some terms with $w_j \approx \widehat{w}_K$ below top K, contributing an opportunity for weight promotion.
- **Lexical provenance**. This gating function naturally blends the idea of preserving original tokens borrowed from the previous studies [12, 40]. Our configuration of β yields $G(w_j, v) \approx 0$ when term $\mathbb{I}(t_j \in v_{\text{orig}})$, leading to no loss penalty in L_{top}. Thus, the trained retrieval model captures this signal to keep the original terms, as original human-provided terms carry explicit intent.

Example. Given a technical query: "quantum computing impact on finance", initial sorted weights are: computing (0.5), portfolio (0.45), finance (0.45), impact (0.3), algorithm (0.3), risk (0.3), quantum (0.24), liquid (0.1).

For HT with a global query threshold of 0.26, any term with a weight below this value is pruned. The final vector is { computing, portfolio, finance, impact, algorithm, risk }. This affects retrieval relevance, only matching documents containing these general terms. The actual specific intent (Quantum) of the original query is lost.

For STop with $K = 4$, it will keep "quantum" because it appears in the original text. It may also yield a negative gradient for "algorithm" and "risk", and possibly boost them because they have the same

weight as $\widehat{w}_K$=0.3 and $G(w_j, v) = 0.5$. In fact, keywords "algorithm" and "risk" are related concepts in option pricing and risk analysis algorithms that quantum computers can potentially speed up. As a result, the model identifies core related keywords and preserves intent with relevant expansions while still demoting "liquid" as a possible contextual noise or an irrelevant term.

Summary and limitation. The learnable soft top-K can predict an optimized number of terms for each vector in a count-adaptive manner, as stated in the last row of Table 2. A weakness of this method is its sensitivity to noise in token importance rankings generated by the pre-trained and refined LLM. Under such distorted rankings, penalizing tokens beyond the soft position K might negatively impact the preservation of useful words, causing brittle cliff behavior under noise. Having a relatively large soft expansion factor b minimizes the impact of this limitation. STop also does not directly target the large number of low-weight terms produced by Lion-SP. This motivates the second component of AdaSparse, PTT, which performs fine-grained filtering of low-weight terms through individualized thresholds.

4.2 PTT: Per-term thresholding

Pruning during document indexing and online query vector generation. PTT prunes a term by setting its term weight $\widetilde{w}_j$ to zero, if its score falls below a certain threshold:

$$\widetilde{w}_j = \begin{cases} w_j, & w_j > \tau_j \\ 0, & w_j \le \tau_j \end{cases} \tag{6}$$

where τ_j is a term-specific threshold learned from the training process discussed below. It can denote either $\tau_{D,j}$, the threshold used for document indexing, or $\tau_{Q,j}$, the threshold used for online query vector generation.

Threshold learning during training. During training, as depicted in Figure 2(a), we use the following thresholding function for both queries and documents for smooth gradient propagation:

$$\widetilde{w}_j = w_j \cdot \sigma(\frac{w_j - \tau_j}{\pi}) \tag{7}$$

- w_j: original weight for term $t_j \in V$ under vocabulary V.
- π controls the steepness of the sigma function curve in approximating pruning, allowing soft gradients during training and hard pruning during inference. We follow the setting of HT with π=0.04.
- τ_j: the threshold for term t_j. Query and document have two distinct sets of term threshold values.

The regularization loss leveraging the above individualized thresholding for queries and documents is defined as follows, generalized from the work of HT [33].

$$L_T = \frac{1}{|V|} \sum_{j \in V} [\log(1 + e^{-\tau_{D,j}}) + \log(1 + e^{-\tau_{Q,j}})].$$

where $\tau_{D,j}$ and $\tau_{Q,j}$ are the thresholds learned for term t_j's of documents and queries, respectively.

Summary and limitation. PTT offers an individualized value-adaptive threshold for pruning to address the weakness of the HT method with finer granularity. However, it still has an important limitation. As stated in Row 5 of Table 2, if the weight distribution of documents for the same term is extremely "flat," too many documents still keep such a term without effective pruning. Our evaluation finds that when we place a large regularization parameter λ_T for using PTT in the Lion-SP's loss function, the sparsity of vectors hardly increase after certain training steps. Compared to PTT, the learnable soft top-K method discussed in Section 4.1 is more effective in addressing the above sparsity resistance issue.

4.3 Multi-objective sparsity regularization

We discuss how per-term thresholding and the learnable soft top-K mechanism are integrated into the model training. For a batch of training queries B, the original loss in Formula(2) is extended as:

$$\mathcal{L} = (\frac{1}{|B|} \sum_{q \in B} L_R) + \lambda_Q L_Q + \lambda_D L_D + \lambda_T L_T + L_{top}. \tag{8}$$

L_R is the rank loss following Lion-SP's setup. The sparsity regularization portion in Formula(8) has multiple learning objectives: FLOPs-based losses L_Q and L_D are the same as the ones in Formula (2) used in Lion-SP. L_T is the regularization term corresponding to PTT, controlled by hyperparameter λ_T. Loss term L_{top} is defined in Formula (5) and introduced by STop. λ_Q, λ_D, λ_T, λ_{qtop} and λ_{dtop} are hyperparameters for component weighting.

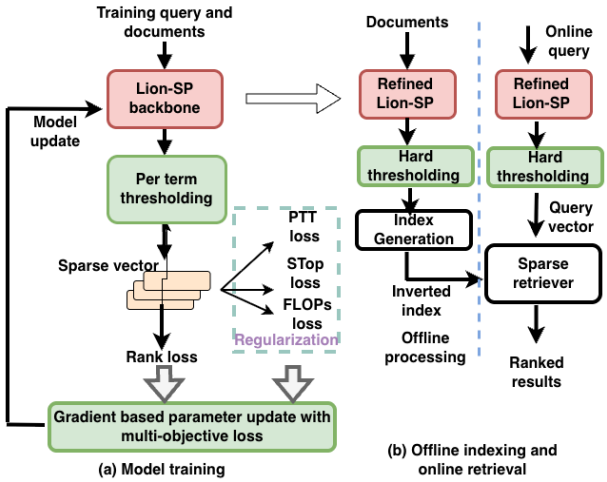


Figure 2: Offline and online processing with AdaSparse

Workflow. Figure 2 illustrates the workflow of the AdaSparse framework. During training, AdaSparse adds two regularization components on top of Lion-SP training. The sparse representation generated by the Lion-SP backbone first goes through the per-term thresholding (PTT) module that attenuates low-weight terms to near-zero using Formula (7). We then compute the soft top-K selection signal described in Section 4.1 and compute the overall loss, which consists of the rank loss, the FLOPs sparsification regularization terms, and two regularization components introduced in this work. The gradients of this loss are then computed, and the trainable model weights are updated accordingly.

During inference, AdaSparse drops terms whose weights fall below the threshold and then follows the same encoding procedure as Lion-SP, without applying any explicit top-K pruning. The encoded sparse document vector will be indexed into an inverted index. The

query vector, which is encoded in the same way, will be used to retrieve the relevant documents from the index.

Table 3: Downward force gravitation on weight w_j by each of the sparsification losses in AdaSparse at a training step

Targeted term weight w_j	Gradient	Effect on w_j
FLOPs with L_Q and L_D		
All terms	$2\overline{w_j}w_j$	Downward force
Soft top-K with L_{top}		
Soft top K ($w_j \geq \widehat{w}_K$)	≈ 0	Minimum impact
Original term $t_j \in v_{orig}$	≈ 0	Minimum impact
Below top K, $w_j < \widehat{w}_K$	$\approx 2w_j$	Downward force
Below top K, $w_j \approx \widehat{w}_K$	$\approx w_j(1-\frac{w_j}{4\gamma})$	Upward for non-small w_j otherwise downward
Per-term thresholding with L_T		
$w_j \geq \tau_{Q,j}$ or $\tau_{D,j}$	0	No update impact
$w_j < \tau_{Q,j}$ or τ_{D_j}	0	$\widetilde{w}_j \approx 0$ by Formula (7)

Gradients for downward weight updating during training. Given a term weight w_j, Table 3 summarizes downward force gradients for targeted term weights of vectors in AdaSparse, yielded from each of the three sparsification techniques.

For FLOPs, let L be either L_Q or L_D and $\overline{w_j}$ is the average weight of term t_j of queries or documents within a training batch:

$$\frac{\partial L}{\partial w_j} = 2\overline{w_j}w_j.$$

For soft top-K, let L be L_{top} scaled down by λ_{qtop} when w_j is a query term or by λ_{dtop} when w_j is a document term.

$$\frac{\partial L}{\partial w_j} = 2w_jG(w_j,v) + w_j^2G(w_j,v)(1-G(w_j,v))\frac{-1}{\gamma}.$$

There are three cases:

- For a term t_j whose weight is above the pivot $\widehat{w}_K$, or for a term that belongs to the original vector before expansion, the gate satisfies $G(w_j,v) \approx 0$. Consequently, the corresponding gradient is approximately zero.
- Otherwise t_j is not in top-K and if $w_j < \widehat{w}_K$,$G(w_j,v) \approx 1$, and

$$\frac{\partial L}{\partial w_j} = w_jG(w_j,v)(2-\frac{w_j(1-G(w_j,v))}{\gamma}) \approx 2w_jG(w_j,v) \approx 2w_j.$$

- If t_j is not in top-K but $w_j \approx \widehat{w}_K$, then $G(w_j,v) \approx 0.5$.

$$\frac{\partial L}{\partial w_j} = w_jG(w_j,v)(2-\frac{w_j(1-G(w_j,v))}{\gamma}) \approx w_j(1-\frac{w_j}{4\gamma}).$$

When $w_j \leq 4\gamma$, $\frac{\partial L}{\partial w_j} \geq 0$. Otherwise $\frac{\partial L}{\partial w_j} < 0$. There is an opportunity for such a term weight below top K to be increased if the rank loss gradient supports. Notice our evaluation sets $\gamma = 0.04$ and when $w_j > 0.16$ and $w_j \approx \widehat{w}_K$, term t_j has a second chance to be weighted more.

For PTT, the gradient with respect to query or document weight w_j is always 0 while this weight is re-scaled at each training step when the corresponding term threshold $\tau_{Q,j}$ or $\tau_{D,j}$ is updated.

Choices of hyperparameters for loss weighting. We consider L_T equally important as rank loss, and thus $\lambda_T = 1$. We let $\lambda_{qtop} = \lambda_Q$ and $\lambda_{dtop} = \lambda_D$ so that the gradients from FLOPs and STop are on a comparable scale. This allows the nonzero gradient contribution from STop to become significant in updating weight w_j.

Synergy of the three techniques and small term weight handling. The tripartite sparsity regulation structure in AdaSparse outperforms standalone methods or bipartite combinations in terms of relevance and/or sparsity through a self-correcting feedback loop discussed as follows. The rank loss suppresses weights that hurt ranking accuracy while preserving or promoting weights that improve it. FLOPs applies a global downward pressure to term weights and is effective at shrinking large noisy activations, but its gradient can become weak for already small weights. Complementary to FLOPs, STop provides a vector-adaptive sparsity signal. As shown in the above gradient analysis, if term weight w_j exceeds the vector's pivot value or $w_j \in v_{orig}$, $G(w_j,v) \approx 0$, STop contributes little additional penalty and the term is mainly governed by the rank loss and FLOPs. Once a term falls below the pivot value, even if **such weight can be small**, its gradient is amplified by the STop. This accelerates the decrease of the weight toward zero. If such a term is important for relevance, the rank loss can counteract this sparsity pressure and promote it back among the soft top-K terms. PTT identifies small term weights and approximates their removal by scaling them down with a sigmoid function, where the scaling is controlled by updated term-specific thresholds.

When the sparsity regularization only uses one or two of the three methods, there are weaknesses as listed in the 4th column of Table 2. AdaSparse addresses them in a collective manner. Table 4 summarizes the names of the main techniques activated for each sparsification scenario, and some problem cases have been discussed in Section 3. For **small term weight** w_j**,** PTT is effective to prune in general but is less effective when there is **a relatively flat distribution** across documents. In that case, FLOPs is also less effective when $0 < w_j < 1$ because its gradient $2\overline{w_j}w_j$ can be small with $0 < \overline{w_j} < 1$. Since $\lambda_{qtop} = \lambda_Q$ and $\lambda_{dtop} = \lambda_D$, STop can contribute the most significant gradient value with its gradient $2w_j$ when $G(w_j,v)$ is close to 1.

Table 4: Main techniques applicable to term weight scenarios

Scenarios	Main techniques
Relevant and irrelevant weights	Rank loss
Large unimportant weights Noisy term weight rankings	FLOPs
Small unimportant weights Diverse term weight distribution	Per-term threshold.
Small weight with flat distribution Excessively long expansion	Soft top K

5 Evaluation Studies

5.1 Settings

Datasets and relevance metrics. The MS MARCO dataset [1, 10] with about 8.8M passages is used for model training. Measured using Huggingface LLaMA-3.2 tokenizer and excluding the beginning special token, the average length of the original queries before expansion in the MS MARCO passage dataset is approximately 8 words, while the average passage length is 76. The performance

evaluation for in-domain search is conducted on its passage Dev set with 6,980 queries, TREC Deep Learning (DL) 2019 set with 43 judged queries, and DL 2020 sets with 53 judged queries. Following the common practice, we report the mean reciprocal rank (MRR@10) for Dev, and report NDCG@10 for TREC DL sets.

We also evaluate zero-shot retrieval performance on the BEIR collection which contains 13 datasets [36] with an average length of original queries ranging between 5 and 245 words, and the average original document length ranging between 14 and 369 words.

Model training and configurations. By default, we use the LLaMA-3.2 with 1 billion parameters as the primary LLM and we also report the performance with 8 billion parameters. We build on the Lion-SP codebase [27] and will release our implementation on GitHub[1]. The model is trained with a combination of contrastive loss (CL) and knowledge distillation (KD), following Lion-SP, and has two steps: 1) We pre-train the LLaMA-3.2 model on the MS MARCO corpus using a masked next-token prediction objective. 2) We then fine-tune the model on MS MARCO passages using teacher-generated scores for KD and hard labels for CL.

Pretraining is conducted on 2 NVIDIA H100 GPUs, with a per device batch size of 32. Finetuning on MS MARCO is conducted on 4 NVIDIA H100 GPUs with 1 epoch. When comparing different loss functions for the Lion-SP refinement, we always start from the same pretrained model, and only change the loss functions and other necessary settings for a fair comparison across different methods.

Sparsity and latency metrics. We report the average lengths of the expanded query and document vectors generated by the AdaSparse and the baselines. Shorter lengths typically imply shorter retrieval latency on average, and thus, query and document lengths serve as a proxy for retrieval time cost. We will study this relationship in more detail and report the actual latency of learned representations when the corresponding inverted indices are executed by a state-of-the-art Seismic retriever [6] at retrieval depth 10 and 1,000. Because Seismic only supports unsafe search, the comparison is performed under approximately matched effectiveness loss relative to safe retrieval. This evaluation is conducted on AMD EPYC 9R45 Processor CPU with 256GB of memory. The latency reported is the average of three runs to minimize variability.

Table 5: Default AdaSparse hyperparameters used

Hyperparameters	Derivation	Value setting
Weights for learning loss components		
λ_Q and λ_D for FLOPs	Follow [39]	$\lambda_Q = 0.05$, $\lambda_D = 0.04$
λ_T for PTT	Follow [33]	$\lambda_T = 1$
λ_{qtop} for STop	$\lambda_{qtop} = \lambda_Q$	$\lambda_{qtop} = 0.05$
λ_{dtop} for STop	$\lambda_{dtop} = \lambda_D$	$\lambda_{dtop} = 0.04$
Parameters for sigmoid gating functions		
Soft expansion limit	Parameter sweeping	$b = 10$ for query $b = 8$ for doc.
Lexical provenance	$\beta = max(\widehat{w}_K)$	$\beta = 2$
Temperature for PTT	Follow [33]	$\pi = 0.04$
Temperature for STop	$\gamma = \pi$	$\gamma = 0.04$

[1]https://github.com/ViViVidam/AdaSparse

Hyperparameter setting. Table 5 lists hyperparameter values used in training the Lion-SP 1B and 8B models with AdaSparse. Column 2 lists how we derive each value. We follow Lion-SP's setup for λ_Q and λ_D and HT's setup for λ_T and π. For β, λ_{qtop}, and λ_{dtop}, our design constrains them as discussed in Sections 4.1 and 4.3. We set $\pi = \gamma$ as both are for temperature-like smoothing in sigmoid gating. For soft expansion factor b, we sweep the hyperparameter value on the MS MARCO passage training data. Section 5.3 gives a sensitivity study when varying b and λ_T.

5.2 A comparison with baselines

A comparison on relevance and sparsity on MS MARCO Dev and BEIR. Table 6 compares the performance of trained Lion-SP using AdaSparse and other baselines. For the baselines, Lion-SP-1B and Lion-SP-8B use official Lion-SP releases with 1B and 8B checkpoints. Lion-SP-1B-repro is our reproduction of model Lion-SP-1B trained with 1 epoch. Lion-SP-8B also uses 1 epoch in training. FLOPs-1B is one variation of Lion-SP that we produce with $\lambda_Q = 0.6$ and $\lambda_D = 0.4$ for queries and documents, respectively. We have tried a number of variations of FLOPs-1B with different loss coefficient parameters for FLOPs loss to force the model to produce the average vector length close to what AdaSparse-1B delivers. L1-FLOPs-1B uses L_1 regularization for each query and keeps FLOPs for documents in training Lion-SP, following [20].

Top-305-1B uses $K = 305$ following [37]. VDR-256-1B keeps top 256 including original words following its setting [40]. MRP-1B listed uses $\alpha = 0.75$ for both documents and queries, which produces a sparsity level close to what the AdaSparse setting delivers. We apply MRP as a post-processing method to the reproduced Lion-SP-1B model. HT-1B uses the recommended HT's configuration [33].

Table 6 shows that AdaSparse produces much shorter sparse vectors while its relevance remains highly competitive to the original Lion-SP model. Compared to Lion-SP-1B-repro, AdaSparse-1B shows negligible performance degradation (≈0% on MS MARCO Dev and 0.74% on BEIR), while reducing query and document lengths to 31% and 26.6% of the original, respectively. Similarly, AdaSparse-8B exhibits less than a 0.3% performance drop on MS MARCO Dev and ≈1.6% degradation on BEIR, with an 82% reduction in query length and a 79.6% reduction in document length.

In general, AdaSparse outperforms other baselines in balancing sparsity and relevance. Among all static pruning methods with top term selection which can easily achieve a target sparsity level, MRP is more competitive than Top-K and VDR in relevance. Top-K and VDR with uniform pruning for all vectors can remove meaningful terms from complex documents, hurting relevance. Even MRP's pruning is vector-specific based on the vector mass, its relevance is still weaker than AdaSparse, especially for BEIR with about 2.3% lower average NDCG@10 due to its non-learnable pruning. Compared to Lion-SP-1B-repro, HT-1B has a 1% MRR degradation on MS MARCO Dev and 1.6% on BEIR. Furthermore, its sparsity is suboptimal: the query length drops by only 9.1%, and its document length is 39.6% longer than that of AdaSparse-1B.

This table also shows that Lion-SP/AdaSparse with 8B parameters compared to 1B can boost 2.2% for MS MARCO Dev, 1% for DL19, 3.2% for DL20, and 0.4% for BEIR. Document vector length changes by only 6%, while query length increases by 30%, from 65 to 85.

Table 6: A comparison of AdaSparse and baselines in relevance and sparsity (average vector length) with LLaMA-3.2

	MS M. MRR	DL19 NDCG	DL20 NDCG	BEIR NDCG	Query Len	Doc Len
Lion-SP-1B [39]	**0.410**	0.747	**0.753**	0.535	293	1250
Lion-SP-1B-repro	0.408	0.753	0.744	**0.545**	208	1052
FLOPs-1B	0.400	0.745	0.743	0.530	130	315
L1-FLOPs-1B	0.404	0.754	0.718	0.531	98	297
Top-305-1B	0.372	0.709	0.690	0.231	85	308
VDR-256-1B	0.398	0.741	0.717	0.493	103	272
MRP-1B (0.75)	0.406	0.748	0.742	0.529	78	**268**
HT-1B	0.404	0.754	0.744	0.536	189	391
AdaSparse-1B	0.408	**0.755**	0.729	0.541	**65**	280
Lion-SP-8B [39]	**0.418**	0.760	0.766	**0.552**	476	1457
AdaSparse-8B	0.417	**0.763**	0.752	0.543	**85**	**297**

Table 7: Latency and storage comparison of trained representations. Latency is measured in milliseconds (ms).

Model	Latency (ms)		MRR@10 (Dev)			Storage
	$k = 10$	$k = 1000$	$k = 10$	$k = 1000$	Safe	(GB)
Lion-SP-1B	1.179 (4x)	9.048 (3.8x)	0.407	0.408	0.410	138 (4.2x)
Lion-SP-8B	1.341 (4.6x)	9.812 (4.1x)	0.415	0.416	0.418	154 (4.7x)
HT-1B	0.613 (2.1x)	3.223 (1.4x)	0.402	0.403	0.404	58 (1.8x)
AdaSparse-1B	0.292 (1x)	2.390 (1x)	0.401	0.406	0.408	33 (1x)
AdaSparse-8B	0.317 (1.1x)	2.863 (1.2x)	0.413	0.415	0.417	34 (1x)

Latency and storage cost. Table 7 shows the latency with retrieval depth 1,000 and storage cost of Lion-SP, AdaSparse and HT. We use a uniform Seismic setting across all the datasets for the same model; however, different models might have different Seismic settings due to the length difference. To run Seismic efficiently, a grid search on its parameter choice for MS MARCO is needed [6]. Our Seismic setting is chosen so that the MRR@10 on the MS MARCO passage Dev set is within 0.002 of safe search. This is intended to align the approximation-induced effectiveness loss across methods, so that the latency comparison reflects end-to-end retrieval efficiency under comparable retrieval quality degradation. For AdaSparse, AdaSparse-8B and HT-1B models, we set Seismic's parameters $\alpha = 0.4$, $\lambda = 4000$, given a similar document sparsity level. We use query cut 8 for HT-1B model, while 6 for AdaSparse and AdaSparse-8B because HT-1B has a longer query length. Column 6 is MRR@10 of safe retrieval, and it is slightly higher than Columns 4 and 5 due to the retrieval approximation of Seismic. The result shows there is a strong correlation between higher sparsity of a trained representation and shorter latency. AdaSparse yields shorter document lengths on average and thus its inverted index space is proportionally smaller while the online latency is much shorter than HT-1B and original Lion-SP.

Detailed Results on MS MARCO and BEIR datasets. Table 8 shows the average query and document lengths for each of 13 BEIR datasets and MS MARCO before and after expansion when using AdaSparse and the original Lion-SP regularization loss (FLOPs). The three columns marked as "Relevance" list MRR@10 for MS MARCO Dev and NDCG@10 for BEIR. The average number listed in the last row is calculated on 13 BEIR datasets, excluding MS MARCO. Latency is in milliseconds. Soft scaling limit during model training is $b = 10$ for queries and 8 for documents. Token counting for lengths is based on the LLaMA-3 tokenizer for all columns. The actual expansion factor listed is the vector length after model expansion divided by the original length for queries and documents, respectively. We include results at the dataset level on BEIR to show that the model behavior is consistent in both out-of-domain and in-domain scenarios. The results in Table 8 suggest the following key observations:

- **AdaSparse vs. Lion-SP with FLOPs only regularization**. AdaSparse shows strong robustness, maintaining high sparsity for expanded document and query vectors in both in-domain and out-of-domain settings. On the BEIR benchmark, AdaSparse reduces the average vector length to 20% of Lion-SP's query length and 25% of its document length.
- **AdaSparse-1B vs. HT-1B**. AdaSparse's MRR@10 for MS MARCO Dev is 1% higher than that of HT-1B, while query length is 2.9x shorter and the document length is 1.4x shorter. Their relevance is comparable on DL'19 and DL'20. AdaSparse's BEIR NDCG@10 is 0.9% higher than that of HT-1B meanwhile the overall sparsity yielded by AdaSparse is much higher than HT-1B. These results demonstrate that our design effectively addresses the weakness of HT-1B in filtering out low-weight terms and managing the score distribution shift. This allows AdaSparse to preserve relevance while achieving a more aggressive overall sparsity.
- **AdaSparse's expansion adaptiveness of query and document lengths on different datasets**. The actual document expansion factors of all datasets are much smaller than the soft limit ($b = 8$). For actual query expansion factors, 11 out of 14 datasets fall into the soft top-K limit of AdaSparse with MS MARCO training ($b = 10$). The other 3 BEIR datasets have generated query lengths 11.7x to 13.5x longer than the original one. This also shows that AdaSparse is adaptive to the actual expansion need. When semantically rich terms are needed for relevance, AdaSparse still expands queries beyond the trained soft limit b. The number of non-zero entries generated by AdaSparse scales along with the input length with dynamic adjustments of the final sparsity level adaptive to the input length. AdaSparse generates the longest length on the ArguAna dataset, as ArguAna input is the longest among all. Meanwhile, AdaSparse generates the shortest vector on NFCorpus, as the input length is the shortest.

5.3 Sensitivity of hyperparameters b and λ_T

The left part of Table 9 shows the impact of varying soft expansion limit factor b for queries from 10 to 5 while keeping soft limit for documents as 8: the query length decreases accordingly, accompanied by a small drop in BEIR NDCG@10 score. Since the document and query encoders are shared, only varying the query limit b also impacts document sparsity in some degree. The right part of Table 9 shows the impact of increasing λ_T from 1 to 6. Increasing λ_T also shortens query and document vectors and similarily overly aggressive sparsification affects relevance.

Comparing the left and right parts of Table 9, decreasing query limit b can yield sparser query vectors than increasing λ_T while having a smaller zero-shot performance drop and a similar MS MARCO relevance. As marked in Row 1 of Table 8 for MS MARCO,

Table 8: Average query length (Q Len) and document length (D Len) on MS MARCO Dev and BEIR datasets. b=10 for queries and 8 for docs. Average row excludes MS MARCO Dev. Ratio in the average is the average number of all the ratios across datasets.

	Original without expansion		LION-SP-1B (FLOPs)			HT-1B (Hybrid Thresholding)			AdaSparse (FLOPs+PTT+STop)		
Dataset	Q Len	D Len	Q Len	D Len	relevance	Q Len	D Len	relevance	Q Len	D Len	relevance
MS MARCO Dev	8	76	293 (36.6x)	1251 (16.5x)	0.410	189 (23.6x)	391 (5.1x)	0.404	65 (8.1x)	280 (3.7x)	0.408
TREC-COVID	15	235	349 (23.3x)	2212 (9.4x)	0.785	179 (11.9x)	765 (3.3x)	0.829	85 (5.7x)	555 (2.4x)	0.817
NFCorpus	5	342	290 (58.0x)	2586 (7.6x)	0.364	190 (38.0x)	910 (2.7x)	0.370	63 (12.6x)	702 (2.1x)	0.371
NQ	10	108	298 (29.8x)	1524 (14.1x)	0.586	186 (18.6x)	543 (5.0x)	0.581	71 (7.1x)	373 (3.5x)	0.579
HotpotQA	23	70	502 (21.8x)	1347 (19.2x)	0.692	222 (9.7x)	505 (7.2x)	0.692	123 (5.3x)	311 (4.4x)	0.698
FiQA-2018	13	169	398 (30.6x)	1740 (10.3x)	0.378	169 (13.0x)	552 (3.3x)	0.378	89 (6.8x)	433 (2.6x)	0.381
ArguAna	245	208	2363 (9.6x)	2220 (10.7x)	0.488	450 (1.8x)	758 (3.6x)	0.487	324 (1.3x)	565 (2.7x)	0.470
Touche-2020	8	369	337 (42.1x)	2005 (5.4x)	0.329	153 (19.1x)	671 (1.8x)	0.318	70 (8.8x)	509 (1.4x)	0.348
DBPedia-entity	7	76	288 (41.1x)	1385 (18.2x)	0.464	174 (24.9x)	525 (6.9x)	0.446	69 (9.9x)	326 (4.3x)	0.452
SCIDOCS	13	225	597 (45.9x)	2282 (10.1x)	0.170	268 (20.6x)	872 (3.9x)	0.169	124 (9.5x)	582 (2.6x)	0.170
FEVER	11	125	672 (61.1x)	1554 (12.4x)	0.870	237 (21.5x)	607 (4.9x)	0.856	148 (13.5x)	386 (3.1x)	0.878
Climate-FEVER	25	125	958 (38.3x)	1554 (12.4x)	0.304	285 (11.4x)	607 (4.9x)	0.249	212 (8.5x)	386 (3.1x)	0.281
SciFact	19	320	1021 (53.7x)	2741 (8.6x)	0.734	321 (16.9x)	997 (3.1x)	0.731	222 (11.7x)	720 (2.3x)	0.727
Quora	12	14	365 (30.4x)	393 (28.1x)	0.791	165 (13.8x)	154 (11.0x)	0.866	82 (6.8x)	85 (6.1x)	0.858
Average	31	184	649 (37.4x)	1811 (12.8x)	0.535	231 (17.0x)	651 (4.7x)	0.536	129 (8.3x)	456 (3.1x)	0.541

Table 9: Impact of hyperparameter changes in AdaSparse-1B

Different b for query					Different λ_T				
Setting	MS M.	Q Len	D Len	BEIR	Setting	MS M.	Q Len	D Len	BEIR
$b = 10$	0.408	65	280	0.541	$\lambda_T = 1$	0.408	65	280	0.541
$b = 8$	0.408	59	273	0.539	$\lambda_T = 3$	0.408	58	257	0.539
$b = 5$	0.407	48	262	0.536	$\lambda_T = 6$	0.408	54	245	0.534

Table 10: Comparison under two sparsity configurations with 1B. Q/D Len denotes average query and document lengths. Relevance columns report MS MARCO Dev MRR@10 and BEIR average NDCG@10, respectively.

Method	Smaller Config.		Default Config.	
	Q/D Len	MS M./BEIR	Q/D Len	MS M./BEIR
AdaSparse	50 / 206	0.409 / 0.537	65 / 280	0.408 / 0.541
FLOPs	100 / 187	0.382 / 0.508	130 / 315	0.400 / 0.530
L1-FLOPs	104 / 221	0.397 / 0.512	98 / 297	0.404 / 0.531
MRP	60 / 204	0.394 / 0.526	78 / 268	0.406 / 0.529

AdaSparse expands 8.1x on average for queries under soft limit 10 and expands 3.7x for documents under limit 8. Even though there is a gap between the actual expansion factor and targeted soft limit b, varying b still gives a much more direct and explicit control of generated vector lengths than tuning other hyperparameters including λ_T. Thus b is chosen as our primary governing parameter to constrain the targeted sparsity while keeping $\lambda_T = 1$ as shown in Table 5.

Two sparsity configurations. Table 10 compares the performance of AdaSparse and few baselines in two sparsity configurations. The default configuration of AdaSparse uses hyperparameters from Table 5 while the smaller configuration is the same except that the soft expansion limit factor is changed as $b = 6$ for queries and $b = 5$ for documents. To match the above two configurations in training Lion-SP with 1B, we adjust the hyperparameters of other baselines for a similar average document length. MRP uses $\alpha = 0.75$ for the default while 0.65 for the smaller configuration. FLOPs uses (λ_q, λ_d) = (0.6,0.4) for the default and (5,4) for the smaller. L1-FLOPs uses (λ_q, λ_d) = (0.005,0.5) for default and (0.006,1.5) for the smaller. The results show that the smaller configuration of AdaSparse retains MS MARCO relevance of the default configuration with an almost identical MRR number while having a modest relevance drop for zero-shot retrieval with BEIR due to more aggressive sparsification. On the other hand, the smaller AdaSparse configuration still outperforms the relevance of other baselines in both smaller and default configurations at a similar or better sparsity level.

Table 11: Ablation studies with different combinations

Performance with 1B	MS MARCO Dev			BEIR
	MRR@10	Q Len	D Len	NDCG@10
FLOPs	0.408	208	1052	0.545
FLOPs + STop	0.409	83	364	0.540
FLOPs + STop + PTT	0.408	65	280	0.541
FLOPs + STop + UTT	0.407	46	225	0.530

5.4 Ablation studies with different sparsification compositions

Table 11 further compares different combinations of sparsification techniques. The first three rows in Table 11 show the impact of adding one sparsification technique incrementally. The result shows that STop can significantly reduce the document and query lengths, while retaining competitive relevance (within 1% degradation), and PTT can further visibly improve sparsity without compromising the relevance. Row 4 replaces PTT with UTT (uniform term thresholding) which maintains one global threshold for all query terms, and one for all document terms. FLOPs+STop+UTT leads to a shorter query length, but with a near 2% NDCG drop for BEIR compared to FLOPs+STop+PTT.

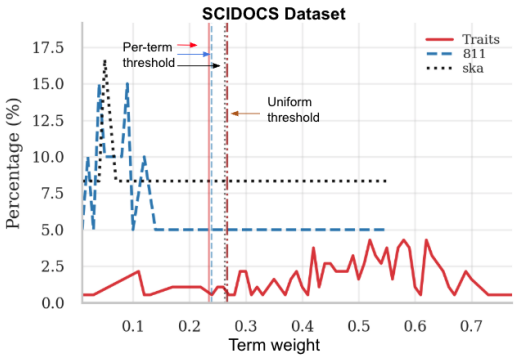


Figure 3: Weight distribution of three terms

Figure 3 provides an example that offers insight into the impact of UTT vs PTT applied to a representation derived by using FLOPs+STop only on SCIDOCs from the BEIR collection. This figure illustrates the weight distribution of three representative terms ("Traits", "811", and "ska") from the popular, the medium-popular, and the unpopular tiers, generated by AdaSparse with FLOPs+STop. Each color represents a different term's distribution, and term percentage is based on the frequency of occurrence in the inverted index. Vertical lines represent one global threshold derived by UTT and three per-term thresholds by PTT. Since the curves of weight distributions vary significantly among different terms, a uniform threshold value is unsuitable. The global threshold value from UTT is larger than all the PTT thresholds, which results in a small gain in sparsity as Table 11 suggests, but incurs a greater relevance penalty.

Table 12: Retrieval latency (ms) of Seismic on MS MARCO.

Doc Length	Query Length			
	9	17	34	69
49	1.274	1.289	1.317	1.490
98	1.750	1.753	1.776	1.764
196	2.299	2.278	2.521	2.275
393	3.278	3.203	3.369	3.182

5.5 Sensitivity of Seismic's retrieval time to query and document lengths

Compared to the sparsity of SPLADEv3 index [22] as shown in Table 13 for MS MARCO, Lion-SP-1B's index with AdaSparse is still 2.7x longer in queries and 1.6x longer in documents. Since the query expansion factor is much larger than that of the document, we investigate whether retrieval latency scales disproportionately in relation to query length.

Table 12 shows the Seismic's latency growth when we control and vary the average document length of an inverted index from 49 to 393 and the query length from 9 to 69. The index with varying sizes is obtained by truncating query and document vectors by keeping a fixed number of top weighted terms in a Lion-SP index trained by AdaSparse for MS MARCO. There is a significant monotonic increase in latency as document lengths grow, whereas the latency remains relatively stable or incurs a modest increase as query length increases. The above asymmetric latency sensitivity of Seismic retriever to the query and document lengths matches well with the outcome of AdaSparse in which expansion is relatively tight for documents (3.7x) and relatively loose for queries (8.1x) for MS MARCO from Table 8.

6 Concluding Remarks

This paper addresses a fundamental computational bottleneck in learned sparse retrieval: the tension between high-dimensional semantic expansion within a massive vocabulary space and efficient inference constraints. It introduces a multi-objective sparsification scheme with learnable soft top-K regularization and per-term thresholding, complementing FLOPs regularization synergistically.

Experimental results with MS MARCO for training and testing, and BEIR datasets for zero-shot retrieval evaluation indicate that AdaSparse-1B uses 4.2x less space and is 3.8x-4x faster in Seismic retrieval than Lion-SP , while retaining relevance within 0.74% difference. Compared to Lion-SP-1B with HT, AdaSparse-1B uses 1.8x less storage and is up to 2.1x faster while having 1% higher relevance. Compared to MRP, AdaSparse-1B delivers better relevance at a similar sparsity level, especially on BEIR with 2.3% higher average NDCG@10. Using LLaMA-3.2 8B significantly boosts MS MARCO Dev performance while the gain for BEIR is modest.

Table 13 compares Lion-SP/AdaSparse with the latest SPLADE version V3 [22] and RepLLaMA [28] to examine the tradeoff using different language models and between sparse and dense representations. SPLADEv3 uses a smaller model BERT with 1.7x shorter document length while Lion-SP with AdaSparse gains relevance significantly on both MS MARCO and BEIR. Lion-SP with AdaSparse has 4.3x smaller index size than RepLLaMA dense representation (34GB vs. 145GB) while RepLLaMA requires much more expensive GPU hardware for speed. Relevance of Lion-SP with AdaSparse is better than RepLLaMA by 1.2% on Dev, and by 2.7% and 3.7% on DL19/DL20, but has 1.5% lower average NDCG@10 on BEIR, which suggests room for further improvement in the future.

Note that like SPLADE, Lion-SP and AdaSparse do not mark titles in MS MARCO training [21]. MRR@10 of Lion-SP on Dev is strong and AdaSparse retains such a level. AdaSparse is designed and evaluated for sparsity optimization with Lion-SP. Our future work is to apply or extend it to learned sparse representations on different LLMs with a large vocabulary space. We will also study the use of different sparse retrievers on latency impact, e.g. [8].

Table 13: A comparison with SPLADE and dense RepLLaMA

Methods	LLM	MM Dev	DL19	DL20	QLen	DLen	BEIR
SPLADEv3 [22]	BERT 110M	0.402	0.723	0.754	24	170	0.517
RepLLaMA dense	LLaMA-2 7B	0.412	0.743	0.725	4K	4K	0.551
Lion/AdaSparse	LLaMA-3.2 8B	0.417	0.763	0.752	85	297	0.543

Acknowledgments. We thank anonymous referees for their valuable comments. This work is supported in part by U.S. NSF IIS-2225942 and a fellowship from Zhu Graduate Support Fund, and has used the computing resource of the NSF's ACCESS program. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. NSF.

References

[1] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. arXiv:1611.09268 [cs.CL] https://arxiv.org/abs/1611.09268

[2] Nikita Balagansky, Yaroslav Aksenov, Daniil Laptev, Vadim Kurochkin, Gleb Gerasimov, Nikita Koriagin, and Daniil Gavrilov. 2025. Train One Sparse Autoencoder Across Multiple Sparsity Budgets to Preserve Interpretability and Accuracy. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 10182–10190. doi:10.18653/v1/2025.emnlp-main.515

[3] Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. LLM2Vec: Large Language Models Are Secretly Powerful Text Encoders. In *First Conference on Language Modeling*. https://openreview.net/forum?id=IW1PR7vEBf

[4] Roi Blanco and Alvaro Barreiro. 2007. Boosting Static Pruning of Inverted Files. In *Proc. of SIGIR* (Amsterdam, The Netherlands) *(SIGIR '07)*. Association for Computing Machinery, New York, NY, USA, 777–778. doi:10.1145/1277741.1277904

[5] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. 2023. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning. *Transformer Circuits Thread* (2023). https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[6] Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. 2024. Efficient Inverted Indexes for Approximate Retrieval over Learned Sparse Representations. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 152–162. doi:10.1145/3626772.3657769

[7] Stefan Büttcher and Charles L. A. Clarke. 2006. A Document-Centric Approach to Static Index Pruning in Text Retrieval Systems. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management* (Arlington, Virginia, USA) *(CIKM '06)*. Association for Computing Machinery, New York, NY, USA, 182–189. doi:10.1145/1183614.1183644

[8] Parker Carlson, Wentai Xie, Rohil Shah, and Tao Yang. 2026. Efficient Sparse Retrieval with Lightweight Superblock Pruning. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '26)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3805712.3809739

[9] David Carmel, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, Yoelle S. Maarek, and Aya Soffer. 2001. Static Index Pruning for Information Retrieval Systems. In *Proc. of SIGIR* (New Orleans, Louisiana, USA) *(SIGIR '01)*. Association for Computing Machinery, New York, NY, USA, 43–50. doi:10.1145/383952.383958

[10] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Fernando Campos, and Ellen M. Voorhees. 2020. Overview of the TREC 2020 Deep Learning Track. *ArXiv* abs/2102.07662 (2020).

[11] Zhuyun Dai and Jamie Callan. 2020. Context-Aware Term Weighting For First Stage Passage Retrieval. *SIGIR* (2020).

[12] Zhen Fan, Luyu Gao, and Jamie Callan. 2023. CSurF: Sparse Lexical Retrieval through Contextualized Surface Forms. In *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval* (Taipei, Taiwan) *(ICTIR '23)*. Association for Computing Machinery, New York, NY, USA, 65–75. doi:10.1145/3578337.3605126

[13] Thibault Formal, C. Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE v2: Sparse Lexical and Expansion Model for Information Retrieval. *ArXiv* abs/2109.10086 (2021).

[14] 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* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2353–2359. doi:10.1145/3477495.3531857

[15] 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* (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 2288–2292. doi:10.1145/3404835.3463098

[16] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List. *NAACL* (2021).

[17] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt,

Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[18] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020. Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation. *ArXiv* abs/2010.02666 (2020).

[19] Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. 2024. Sparse Autoencoders Find Highly Interpretable Features in Language Models. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=F76bwRSLeK

[20] Carlos Lassance and Stéphane Clinchant. 2022. An Efficiency Study for SPLADE Models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2220–2226. doi:10.1145/3477495.3531833

[21] Carlos Lassance and Stephane Clinchant. 2023. The Tale of Two MSMARCO - and Their Unfair Comparisons. In *Proceed. of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23)*. ACM, New York, NY, USA, 2431–2435.

[22] Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. 2024. SPLADE-v3: New baselines for SPLADE. arXiv:2403.06789 [cs.IR] https://arxiv.org/abs/2403.06789

[23] Carlos Lassance, Simon Lupart, Hervé Déjean, Stéphane Clinchant, and Nicola Tonellotto. 2023. A Static Pruning Study on Sparse Neural Retrievers. In *Proc. of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 1771–1775.

[24] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.

[25] Ruoxuan Li, Xiaoyao Zhong, Jiabao Jin, Peng Cheng, Wangze Ni, Zhitao Shen, Wei Jia, Xiangyu Wang, Heng Tao Shen, and Jingkuan Song. 2026. SINDI: an Efficient Index for Approximate Maximum Inner Product Search on Sparse Vectors. ICDE 2026 and arXiv:2509.08395 [cs.DB] https://arxiv.org/abs/2509.08395

[26] Jimmy Lin and Xueguang Ma. 2021. A Few Brief Notes on DeepImpact, COIL, and a Conceptual Framework for Information Retrieval Techniques. *CoRR* abs/2106.14807 (2021). arXiv:2106.14807 https://arxiv.org/abs/2106.14807

[27] LION. 2025. https://github.com/HansiZeng/scaling-retriever. (2025).

[28] Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2024. Fine-Tuning LLaMA for Multi-Stage Text Retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 2421–2425. doi:10.1145/3626772.3657951

[29] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning Passage Impacts for Inverted Indexes. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 1723–1727. doi:10.1145/3404835.3463030

[30] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. 2017. Faster BlockMax WAND with Variable-sized Blocks. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Shinjuku, Tokyo, Japan) *(SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 625–634. doi:10.1145/3077136.3080780

[31] Biswajit Paria, Chih-Kuan Yeh, Ian E.H. Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. 2020. Minimizing FLOPs to Learn Efficient Sparse Representations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SygpC6Ntvr

[32] Seongwan Park, Taeklim Kim, and Youngjoong Ko. 2025. Decoding Dense Embeddings: Sparse Autoencoders for Interpreting and Discretizing Dense Retrieval. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 26479–26496. doi:10.18653/v1/2025.emnlp-main.1345

[33] Yifan Qiao, Yingrui Yang, Shanxiu He, and Tao Yang. 2023. Representation Sparsification with Hybrid Thresholding for Fast SPLADE-based Document Retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 2329–2333. doi:10.1145/3539618.3592051

[34] Yuval Reif, Guy Kaplan, and Roy Schwartz. 2025. Vocab Diet: Reshaping the Vocabulary of LLMs with Vector Arithmetic. arXiv:2510.17001 [cs.CL] https://arxiv.org/abs/2510.17001

[35] Tao Shen, Xiubo Geng, Chongyang Tao, Can Xu, Xiaolong Huang, Binxing Jiao, Linjun Yang, and Daxin Jiang. 2023. LexMAE: Lexicon-Bottlenecked Pretraining for Large-Scale Retrieval. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=PfpEtB3-csK

[36] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. 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)*. https://openreview.net/forum?id=wCu6T5xFjeJ

[37] Jheng-Hong Yang, Xueguang Ma, and Jimmy Lin. 2021. Sparsifying Sparse Representations for Passage Retrieval by Top-k Masking. *CoRR* abs/2112.09628 (2021). arXiv:2112.09628 https://arxiv.org/abs/2112.09628

[38] Hamed Zamani, Mostafa Dehghani, William Bruce Croft, Erik G. Learned-Miller, and J. Kamps. 2018. From Neural Re-Ranking to Neural Ranking: Learning a Sparse Representation for Inverted Indexing. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (2018).

[39] Hansi Zeng, Julian Killingback, and Hamed Zamani. 2025. Scaling Sparse and Dense Retrieval in Decoder-Only LLMs. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Padua, Italy) *(SIGIR '25)*. Association for Computing Machinery, New York, NY, USA, 2679–2684. doi:10.1145/3726302.3730225

[40] Jiawei Zhou, Xiaoguang Li, Lifeng Shang, Xin Jiang, Qun Liu, and Lei Chen. 2024. Retrieval-based Disentangled Representation Learning with Natural Language Supervision. arXiv:2212.07699 [cs.CL] https://arxiv.org/abs/2212.07699