Hypencoder: Hypernetworks for Information Retrieval

Julian Killingback
jkillingback@cs.umass.com
University of Massachusetts Amherst
Amherst, MA, USA

Hansi Zeng
hzeng@cs.umass.edu
University of Massachusetts Amherst
Amherst, MA, USA

Hamed Zamani
zamani@cs.umass.edu
University of Massachusetts Amherst
Amherst, MA, USA

ABSTRACT

The vast majority of retrieval models depend on vector inner products to produce a relevance score between a query and a document. This naturally limits the expressiveness of the relevance score that can be employed. We propose a new paradigm, instead of producing a vector to represent the query we produce a small neural network which acts as a learned relevance function. This small neural network takes in a representation of the document, in this paper we use a single vector, and produces a scalar relevance score. To produce the little neural network we use a hypernetwork, a network that produce the weights of other networks, as our query encoder or as we call it a Hypencoder. Experiments on in-domain search tasks show that Hypencoder is able to significantly outperform strong dense retrieval models and has higher metrics then reranking models and models an order of magnitude larger. Hypencoder is also shown to generalize well to out-of-domain search tasks. To assess the extent of Hypencoder's capabilities, we evaluate on a set of hard retrieval tasks including tip-of-the-tongue retrieval and instruction-following retrieval tasks and find that the performance gap widens substantially compared to standard retrieval tasks. Furthermore, to demonstrate the practicality of our method we implement an approximate search algorithm and show that our model is able to search 8.8M documents in under 60ms.

1 INTRODUCTION

Efficient neural retrieval models are based on a bi-encoder (or two tower) architecture, in which queries and documents are represented separately using either high-dimensional sparse [17, 18, 75] or relatively low-dimensional dense vectors [19, 32, 33, 73, 76]. These models use simple and light-weight similarity functions, e.g., inner product or cosine similarity, to compute the relevance score for a given pair of query and document representations. We demonstrate theoretically that inner product similarity functions fundamentally limit the types of relevance that retrieval models can express. Specifically, we prove that there is always a set of relevant documents which cannot be perfectly retrieved regardless of the query vector and specific encoder model.

Motivated by this theoretical argument, we introduce a new category of retrieval models that can capture complex relationship between query and document representations. Building upon the hypernetwork literature in machine learning [22, 59, 66], we propose Hypencoder–a generic framework that learns a query-dependent multi-layer neural network as a similarity function that is applied to the document representations. In more detail, Hypencoder applies attention-based hypernetwork layers, called hyperhead layers, to the contextualized query embeddings output by a backbone transformer encoder. Each hyperhead layer produces the weight and bias matrices for a neural network layer in the query-dependent similarity network, called the q-net. The q-net is then applied to each document representation, which results in a scalar relevance score. We demonstrate that the Hypencoder framework can be optimized end-to-end and can be used for efficient retrieval from a large corpus. Specifically, we propose a graph-based greedy search algorithm that approximates exhaustive retrieval using Hypencoder while being substantially more efficient.

We conduct extensive experiments on a wide range of datasets to demonstrate the efficacy of Hypencoder. We demonstrate that our implementation of Hypencoder for single vector document representations outperforms competitive single vector dense and sparse retrieval models on MS MARCO [49] and TREC Deep Learning Track data [10, 13], in addition to complex retrieval tasks, such as TREC DL-Hard [43], TREC Tip-of-the-Tongue (TOT) Track [3], and the instruction following dataset FollowIR [69]. Across these benchmarks Hypencoder demonstrates consistent performance gain across experiments. Note that using the proposed approximation approach, retrieval from MS MARCO [49] with approximately 8.8 million documents only takes an average of 59.6 milliseconds per query on a single NVIDIA L40S GPU.

A main advantage of hypernetworks in machine learning is their ability to learn generalizable representations. To demonstrate that Hypencoder also inherits this generalization quality, we evaluate our model under various domain adaptation settings: (1) adaptation to question answering datasets in biomedical and financial domains, and (2) adaptation to other retrieval tasks, including entity and argument retrieval, where Hypencoder again demonstrates superior performance compared to the baselines.

We believe that these performance gains are just the by-product of our main contributions; Hypencoder introduces a new way to think about what retrieval and relevance functions can be, it opens a new world of possibilities by bridging the gap between neural networks and retrieval similarity functions. We believe Hypencoder is especially important at this time given the new demands for longer and more complex queries brought on by the widespread usage of large language models and it is our belief that Hypencoder represents an important step towards this goal. To help facilitate this goal we will open source all our code for training, retrieval, and evaluation.[1]

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

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

[1] Available at https://github.com/jfkback/hypencoder-paper

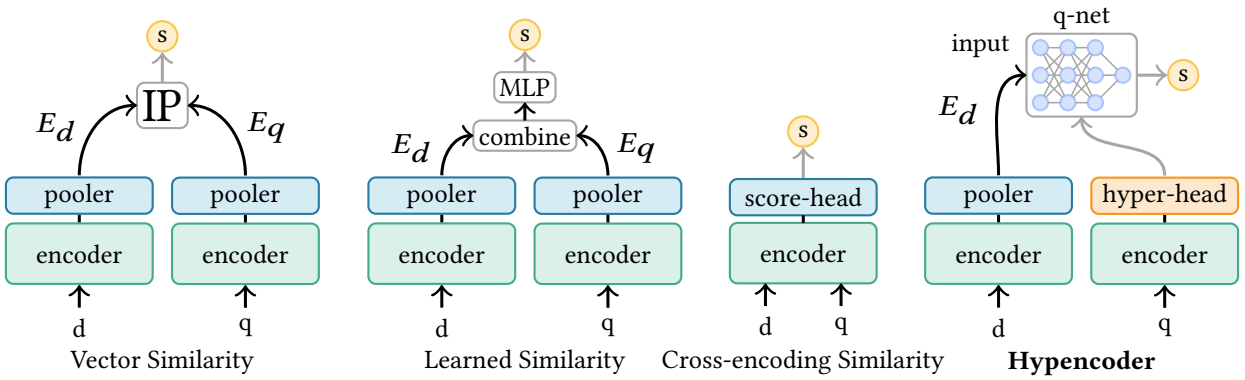

Figure 1: Overview comparing Hypencoder to existing retrieval and reranking paradigms. Gray arrows indicate the arrow does not represent an entity, it is the same as what it points to, in contrast black arrows do indicate a unique entity which is always labeled.

2 RELATED WORK

Vector Space Models. Vector-based models that use sparse vectors have existed for decades, with each index representing a term in the corpus vocabulary. Document-query similarity is computed using measures like l_2 distance, inner product, or cosine similarity, with term weighting methods such as TF-IDF being a substantial focus to improve performance. With the emergence of deep neural networks, focus shifted to learning representations for queries and documents. SNRM by Zamani et al. [75] was the first deep learning model to retrieve documents from large corpora by learning latent sparse vectors. Following works leveraged pretrained transformer models like BERT [16] using single dense vector representations [32]. Recent improvements have focused on training techniques including self-negative mining [52, 53, 73, 77], data augmentation [38, 53], distillation [26, 37, 38], corpus-pretraining [19, 30, 40, 71], negative-batch construction [28] and curriculum learning [39, 76]. Alternative approaches include ColBERT [33], which uses multiple dense vectors, and SPLADE [18], which revisits sparse representations using pretrained masked language models.

Though these methods vary substantially, they all share a fundamental commonality, that relevance is based on an inner product (or in some cases cosine similarity). We believe that this is a significant limitation of these methods and one which hampers the performance of these models on complex retrieval tasks. Our method circumvents this limitation by learning a query-dependent small neural network that is fast enough to run on the entire collection (or used in an approximate way; see Section 3.6 for details).

Learned Relevance Models. Light-weight relevance models using neural networks have demonstrated improved retrieval performance compared to simple methods like inner products. Early iterations came in the form of learning-to-rank models [8, 9] which use query and document features to produce relevance scores for reranking. While these models traditionally used engineered features, more recent approaches adopted richer inputs. For instance, MatchPyramid [51] and KNRM [72] use similarity matrices between non-contextualized word embeddings, while Duet [47, 48] combines sparse and dense term features in a multi-layer perceptron. DRMM [74] utilized histogram features as input to neural networks for scoring. Since the advent of BERT [16], focus has shifted to making transformer models more efficient, such as PreTTR [42] which separately precomputes query and document hidden states. Recently, LITE [31] extended ColBERT's similarity using column-wise and row-wise linear layers for scoring.

In the recommender systems community, learned similarity measures have been widely used [24, 25]. The common usage of neural scoring methods in recommendation has inspired research into efficient retrieval with more learned scoring signals. For instance, BFSG [62] supports efficient retrieval with arbitrary relevance functions by using a graph of item representations and a greedy search strategy over nodes of the graph. A recent improvement on BFSG uses the scoring models gradient to prune directions that are unlikely to have relevant items [79]. Other works make use of queries to form a query-item graph to produce more informative neighbors [61].

Our work differs from these works in one major way, we do not have a query representation and document representation thus our method requires no combination step, instead we produce a query-conditioned neural network for each query and directly apply this to the document representation. This approach can reduce the similarity network's size and does not require choosing between inference speed and larger query representations. Furthermore the flexibility of our framework means we can replicate any existing learned relevance model as discussed in Section 3.7. On a broader note there has been surprisingly little work on neural based scoring for full-scale retrieval, especially in the modern era of transformer based encoders. We hope our work can be a useful foundation and proof-of-concept for future work in this area.

Hypernetworks. Hypernetworks also known as hypernets are neural networks which produce the weights for other neural networks. The term was first used by Ha et al. [22] who demonstrated the effectiveness of hypernetworks to generate weights for LSTM networks. Since then, hypernetworks have been used in a variety of ways including neural architecture search [78], continual learning

[66], and few-shot learning [57, 59] to name a few. Generally, hypernetworks take a set of input embeddings that provide information about the type of task or network where the weights will be used. These embeddings are then projected to the significantly larger dimension of the weights of the "main" network. As the outputs of most hypernetworks are so large the hypernetworks themselves are often very simple such as a few feed-forward layers in order to keep computation feasible. Our case is unique in that our hypernetwork, the Hypencoder, is much larger than the small scoring network which we call q-net (i.e. the "main" network). Additionally, to the best of our knowledge, this paper represents the first work to explore hypernetworks for first stage retrieval.

3 HYPENCODER

Neural ranking models can be generally categorized into early-interaction and late-interaction models [14, 20, 27, 33, 34, 50, 68]. Currently, the most common implementation of early-interaction models is in the form of *cross-encoders* (Figure 1 (second from the left)), where the query text q and document text d are concatenated (together with some predefined tokens or templates) and fed to a transformer network that learns a joint representation of query and document and finally produces a relevance score. The joint representation prevents these models from being able to precompute document representations, thus they cannot be used efficiently on large corpora [21, 46].

The most popular implementation of late-interaction models follows a *bi-encoder* (or two tower) network architecture (Figure 1 (left)), where query and document representations are computed separately and a scoring function is used to estimate the relevance score. Formally, let $E_q \in \mathbb{R}^{n \times h}$ denote the representation learned for query q consisting of n h-dimensional vectors. Similarly, $E_d \in \mathbb{R}^{m \times h}$ denotes the representation learned for document d consisting of m vectors of the same dimensionality. The relevance score between q and d is computed as follows:

$$\psi(E_q, E_d) \tag{1}$$

where $\psi : \mathbb{R}^{n \times h} \times \mathbb{R}^{m \times h} \rightarrow \mathbb{R}$ denotes the scoring function.

In order to take advantage of efficient indexing techniques, such as an inverted index in the case of sparse representations [18, 75] or approximate nearest neighbor (ANN) search in the case of dense representations [32], many existing works use pooling techniques to obtain a single vector representation for each query and document and then employs simple and light-weight scoring functions, such as inner product or cosine similarity. There also exist more expensive methods that do not use pooling and perform such light-weight scoring functions at the vector level and then aggregate them, such as the maximum inner product similarity used in ColBERT [33].

On the Limitations of Linear Similarity Functions (e.g., Inner Product). We believe the simple similarity functions used by existing bi-encoder models are not sufficient for modeling complex relationships between queries and documents. These functions inherently limit retrieval models to judge relevance in a way that can be represented by an inner product. Furthermore, it has been shown that the ability to compress and reconstruct information is correlated with the size, and thus complexity, of neural models [15]. This result indicates that using a relevance function as simple as an inner product likely reduces the amount of information that can be stored in a fixed representation size. These factors explain why state-of-the-art dense retrieval models continue to underperform cross-encoder models, in terms of retrieval quality [36]. In the following, we show the limitations of inner products (as a linear similarity function) by theoretically demonstrating the impossibility of inner products to produce perfect rankings for some queries, regardless of the method used to create the query and document embeddings.

Let C denote a corpus of N documents, each being represented by an h-dimensional vector. A perfect ranking of documents in C for a provided query is a ranking where all relevant documents are ranked above all non-relevant documents. According to Radon's Theorem [54], any set of $h+1$ document vectors with h dimensions can be partitioned into two sets whose convex hulls intersect. An important application of Radon's Theorem is in calculating the Vapnik–Chervonenkis (VC) dimension [64] of h-dimensional vectors with respect to linear separability. For any $h+2$ vectors, the two subsets of a Radon partition cannot be linearly separated. In other words, for $N > h+1$, there exists at least one group of documents that is not linearly separable from the rest. In the real world, since $N \gg h+1$, there are indeed many such non-separable subsets. If any two of these subsets contain all the relevant documents for a query, then no linear similarity function can perfectly separate relevant from irrelevant documents. This includes inner product similarity and guarantees that, for some query, there will be an imperfect ranking.

To overcome these limitations with inner product similarity we use a multi-layer neural network with query-conditioned weights as our similarity measure. As neural networks are universal approximators [29], Hypencoder's similarity function can express far more complex functions than those expressed by inner products. A related alternative approach with the same benefits takes the query and document representations, combines them (e.g., through concatenation or similarity matrices), and feeds them to a neural network to serve as a similarity function (Figure 1 (second from the right)). However, this approach suffers from the following shortcomings: (1) query and document representations now need to be combined before scoring – adding latency proportional to the complexity of the method used to combine them; (2) having separate query and document representations increases the input dimension to the neural network further increasing latency; (3) for efficiency reasons, the query representation is often pooled or compressed before being input into network which reduces the information the model receives. Hypencoder addresses these shortcomings. Since the query is directly encoded as the neural network's weights no concatenation or other form of combining inputs is needed, the document representation can be directly input to the scoring network. This, in addition to the reduced network size from having only document representations as input, allows for a substantial latency improvement. Further, as Hypencoder produces a query-specific neural network, every weight can be used to store query-related information without any need for compression or additional overhead. Lastly, we show in Section 3.7 that existing learned relevance methods can be exactly replicated by Hypencoder with the additional flexibility of learning query-specific weights when desirable.

3.1 Hypencoder Overview

An overview of our model is depicted in Figure 1 (right); it represents a new category of models that sit between a cross-encoder and a bi-encoder model. Like a bi-encoder model, our method computes the query and document representations separately, but unlike most existing retrieval methods, our method allows for more complicated matching signals like those present in cross-encoder models. Following existing methods, we have a query encoder and a document encoder. When a document d is input into to the document encoder, we obtain a representation similar to existing encoder models, namely a set of one or more vectors $E_d \in \mathbb{R}^{m \times h}$ that represent the document's content, where m is the number of vectors and h is the dimension of the vectors. Though we focus on vectors in this work, in theory, the representation can be anything a neural network can output.

Now comes our unique contribution that allows our method to consider more complex similarity signals. Given the query q, the query encoder Φ first produces a set of contextualized embeddings in a similar way to existing encoder models which we will call $E_q \in \mathbb{R}^{n \times h}$, where n is the number of embeddings and h is the dimension of the embeddings. At this point while existing methods apply a simple pooling mechanism, our query encoder instead uses a hyper-head. *The hyper-head takes E_q and produces a set of matrices and vectors that are then used as the weights and biases for a small neural network which we coin the q-net.* The q-net is a query-dependent function for estimating relevance scores for each document, meaning each q-net is unique to the query that created it, unlike existing neural scoring methods which use a shared set of weights for all queries. To find the relevance of a document, the document representation E_d is passed as input to the q-net which outputs the relevance score.

Hypencoder is a generic framework which allows direct application of existing paradigms from neural retrieval and, more broadly, machine learning. For example, Hypencoder could easily work with multiple vectors similar to existing multi-vector models, e.g., [33], or use training routines popularized in dense retrieval, e.g., [38, 52, 73, 76]. As an initial exploration, this paper focuses on showing the efficacy of Hypencoder without additional complexity and thus uses a single vector document representation and no complex training recipes.

3.2 Query and Document Encoders

The Hypencoder framework is generic and can be applied to any implementation of query and document encoders. In this work, we use pretrained transformer-based encoder models commonly used in the recent neural network literature. Specifically, we use a pretrained BERT base model [16] for encoding queries and documents. Even though Hypencoder can operate on all token representations produced for each document this work focuses on a single vector representation of documents, which is more efficient in terms of query latency, memory requirements, and disk usage. To do so, we can either use the contextualized embedding representing the `[CLS]` token or take the mean of all the contextualized embeddings for all the non-pad input tokens. Empirically, we found that using the `[CLS]` token performs better. Therefore, the document representation produced by the encoder is a single vector with 768 dimensions, i.e., the same as BERT's output dimensionality. We refer to it as $E_d \in \mathbb{R}^{m \times h}$, where $m = 1$ in our setting.

Since Hypencoder only uses the contextualized-query-token representations once to produce the q-net, it can skip pooling tokens without adding much cost. Therefore, we use all non-pad-token representations produced by the query encoder as the intermediate representation of the queries, denoted by $E_q \in \mathbb{R}^{n \times h}$, where n is the number of tokens in the query q and h is the embedding dimensionality ($h = 768$ in BERT).

3.3 The Hyperhead Layers

The method to transform E_q into the weights and biases for the q-net is performed by the hyperhead layers and is completely flexible. During our experimentation, we tried two mechanisms to do this transformation as well as many minor variants and found them all to have stable training, which suggests the Hypencoder framework is robust to the exact hyperhead layer implementation. Though we tried two approaches, we settled on one for the final set of experiments in this paper which we will now describe.

For improved clarity, we focus only on the weight creation process as the biases are created in the exact same way. The contextualized query embeddings $E_q \in \mathbb{R}^{n \times h}$ produced by the query encoder are independently transformed by l hyperhead layers, each of which corresponds to a layer in the q-net. Each hyperhead layer converts the embeddings E_q into key and value matrices:

$$K_i^q = \theta_{K_i} \times [E_q; 1] \qquad V_i^q = \theta_{V_i} \times [E_q; 1] \tag{2}$$

where $\theta_{K_i}, \theta_{V_i} \in \mathbb{R}^{h \times h}$ denote learnable parameters for constructing key and value matrices. In the above equation, the embedding matrix E_q is concatenated with a column of all ones (i.e., $[E_q; 1]$) to model both weight multiplication and bias addition.

Each key matrix K_i and value matrix V_i will be used for the creation of the weights in the i^{th} layer of the q-net. With the keys and values in hand, single-head scaled-dot-product attention [65] is performed using a query matrix $Q_i \in \mathbb{R}^{r \times h}$ where r is the layer dimensionality in the i^{th} layer of q-net. In our case, all of the weights except the last layer are square matrices, making $r = h$. Each Q_i is a set of learnable embeddings, similar to those used as input tokens for transformer models. Hence, the hidden layer representation $H_i \in \mathbb{R}^{r \times h}$ is then computed as follows:

$$H_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{h}}\right) V_i \tag{3}$$

A ReLU activation [1] is then applied to each H_i followed by layer normalization [4]. Next a point-wise feed-forward layer is applied to produce $\widehat{H}_i^q$:

$$\widehat{H}_i^q = \theta_{W_i} \text{L-Norm}\left(\text{ReLU}(H_i)\right) + \theta_{b_i} \tag{4}$$

where L-Norm denotes layer normalization. Note that each weight in q-net has a unique θ_{W_i} and θ_{b_i}. There are no learnable parameters in layer normalization.

The final operation to get the i^{th} weight W_i^q for q-net is:

$$W_i^q = \widehat{H}_i^q + \theta_{H_i} \tag{5}$$

where $\theta_{H_i} \in \mathbb{R}^{r \times h}$ is the same size as $\widehat{H}_i^q$ and acts as a base weight which allows the model to learn universal (i.e., query-independent) patterns that are applicable for all queries.

The process for the bias vectors is identical except the query matrix used in the attention operation $Q_i \in \mathbb{R}^{r \times h}$ has $r = 1$ as there is only a single column in the output.

3.4 The q-net Network

Weights and biases produced by the hyperhead layers are not by themselves a neural network. They need a certain arrangement and additional components (e.g. non-linearity). This is where the Hypencoder's q-net converter comes in. The converter knows the architecture of the q-net and given the weights and biases from the hyperhead layers, it produces a callable neural network object which takes as input the document representation E_d.

It is worth highlighting that because the q-net's architecture is not strictly tied to how the hyperhead layer produces the weights and biases, it is simple to modify the architecture of the q-net. All the hyperhead layers need to know is how many weights and biases are needed and what shape they should be.

In our experiments, we use a simple feed-forward architecture for the q-net. The output x_{i+1}^d for the input x_i^d at a given layer i is given by:

$$x_{i+1}^d = \text{L-Norm}\left(\text{ReLU}(W_i^q(x_i^d) + b_i^q)\right) + x_i^d \tag{6}$$

where L-Norm represents a layer normalization without learnable parameters and the addition of x_i^d is a residual connection. No residual connection is applied before the final layer (i.e., layer l). The layer in Equation (6) is repeated l times. Finally, a relevance score is produced using a linear projection layer with an output dimensionality of 1.

3.5 Training

Training Hypencoder is no different from training a bi-encoder as it shares the same core components, i.e., a query encoder and document encoder. The only difference is instead of using an inner product to find the similarity the q-net is applied to the document representations. Thus, our contributions are solely the architecture and not a specific training technique. In this paper we employ a simple distillation training setup, for more details see Section 4.2

3.6 Efficient Retrieval using Hypencoder

Being able to perform efficient retrieval is crucial for many real-world search scenarios where an exhaustive search is not feasible. For Hypencoder models, there is a clear parallel to dense models as both represent documents as dense vectors, but the differences between Hypencoder and dense models make it unclear whether the same efficient search techniques will work. For instance, it is clear that due to the linear nature of inner products, similar document vectors are likely to have similar inner products with a query vector; in the case of Hypencoder this assumption may not hold true as the non-linear nature of the Hypencoder scoring function could mean small differences in the input vector produce significant differences in the output score.

To study the extent to which Hypencoder's retrieval can be approximated for efficient retrieval, we developed an approximate search technique based loosely on navigating small world graphs [35, 45]. In the index stage we construct a graph where documents are nodes connected to their neighbors by edges. We use L_2 distance between document embeddings similar to [62].

After constructing the document graph, approximate search is performed following Algorithm 1. In brief, a set of initial candidate documents $\tilde{C}$ is selected at random, these candidates are scored with the q-net (line 5) and in lines 16-19 the best *nCandidates* and their neighbors become the next candidates. In lines 12-15, the top scoring candidates are added to T–a set which stores the k best scoring documents so far. The algorithm terminates when one of three conditions is met: (1) the number of iterations equals *maxIter*; see line 4, (2) there are no more candidates; see line 4, or (3) no new documents are added to T at a given step; see line 8. We also consider an option without the final termination condition which we call *without early stopping*. As the number of operations is dependent on the number of initial candidates $|\tilde{C}|$, the running time is not tied to the number of documents, resulting in a run time complexity of $O(|\tilde{C}| + nCandidates \cdot maxIter)$.

With this algorithm, we found that Hypencoder is able to significantly increase retrieval speed without a large loss in quality. See the results in Section 4.4.4.

Algorithm 1 Hypencoder Efficient Search

Input: q-net q, #NN to return k, initial candidates $\tilde{C}$, # candidates to explore every iteration *nCandidates*, *maxIter*
Output: k closest neighbors to q

1: $v \leftarrow \tilde{C}$ ▷ set of visited elements
2: $T \leftarrow \{-\infty\}$ ▷ Stores top k nearest neighbors to q at any given time
3: $i \leftarrow 0$ ▷ Current iteration
4: **while** $|\tilde{C}| > 0$ and $i <$ *maxIter* **do**
5: $c \leftarrow$ find top *nCandidates* values in $\tilde{C}$ using q
6: $f \leftarrow$ get lowest scoring element from T
7: **if** $\max_{\hat{c} \in c} \hat{c} < f$ **then**
8: **break** ▷ all candidates are worse than T so stop now
9: $\tilde{C} \leftarrow \{\}$ ▷ Reset $\tilde{C}$
10: **for** each $e \in c$ **do**
11: $f \leftarrow$ get lowest scoring element from T
12: **if** $q(e) > f$ or $|T| < k$ **then**
13: $T \leftarrow T \cup e$
14: **if** $|T| > k$ **then**
15: $T \leftarrow T \setminus \{f\}$
16: **for** each $n \in$ NEIGHBORS(e) **do**
17: **if** $n \notin v$ **then**
18: $\tilde{C} \leftarrow \tilde{C} \cup n$
19: $v \leftarrow v \cup n$
20: $i \leftarrow i + 1$
21: **return** T

3.7 Comparison to Existing Neural IR Methods

We argue that Hypencoder can exactly reproduce existing neural ranking models. Let us start by formalizing the main components of existing neural methods: (1) a query representation E_q, (2) a

document representation E_d, (3) some combination function $f_c(\cdot,\cdot)$, and (4) the final neural network that produces a score $f_s(\cdot)$. In comparison, Hypencoder does not have an $f_c(\cdot,\cdot)$ as the q-net takes E_d as its only input. We will now demonstrate that Hypencoder can exactly replicate any neural retrieval method that has the components above. The first step is to include E_q and $f_c(\cdot,\cdot)$ in the q-net. This allows the q-net to exactly produce the input to $f_s(\cdot)$ that the existing neural retriever used. Next we reproduce the neural function from $f_s(\cdot)$ with query-dependent weights in the q-net. When E_d is input to the q-net, all the original components of the existing neural retrieval model are present and thus the score can be exactly replicated. There is one difference, which is the weights of $f_s(\cdot)$ are query-dependent. However, this can be remedied in two ways: (1) shared weights can be used for all queries exactly replicating the original neural method (2) the weights for f_c can have a common non-query-dependent base weight, similar to our implementation (see details in Section 3.3), this way if there is no benefit using query-dependent weights the shared weight can be used, but if there is additional benefit the model and optimizer can learn to take advantage of it. Thus, Hypencoder can not only exactly replicate all existing neural retrieval methods it also allows the model to dynamically leverage query-dependent weights when the model determines they are beneficial.

4 EXPERIMENTS

4.1 Datasets

4.1.1 Training Dataset. The dataset used for training our models is the training split of the MSMARCO passage retrieval dataset [49] which contains 8.8M passages and has 503K training queries with at least one corresponding relevant passage. The queries in the MSMARCO training set are short natural language questions asked by users of the Bing search engine.

To create the training pairs, we first retrieved 800 passages for every query using an early iteration of Hypencoder. From these, we sampled 200 passages — the top 100 passages and another 100 randomly sampled from the remaining 700 passages. These query-passage pairs were then labeled using the MiniLM cross-encoder[2] from the Sentence Transformers Library [55].

4.1.2 Validation Dataset. For validating and parameter tuning, we use the TREC Deep Learning (DL) 2021 [11] and 2022 passage task [11, 12]. As the passage collection for TREC DL '21 and '22 is large and we wanted validation to be fast we created a subset with only passages in the QREL files.

4.1.3 Evaluation Datasets. Our evaluation explores retrieval performance in three different areas: in-domain performance, out-of-domain performance, and performance on hard retrieval tasks.

For in-domain performance, we use the MSMARCO Dev set [49], TREC Deep Learning 2019 [13], and TREC Deep Learning 2020 [10]. The MSMARCO Dev set contains around 7k queries with shallow labels, the majority of queries only have a single passage labeled as relevant. This collection uses queries from the same distribution as the training queries making it a clear test of the in-domain performance. On this dataset we report the standard evaluation metrics: MRR and Recall@1000. The TREC Deep Learning 2019 and 2020 datasets have a similar query distribution to MSMARCO Dev but feature far fewer queries, i.e., 97 queries combined. The lower number of queries is compensated by far deeper annotations with every query having several annotated passages.

To assess out-of-domain performance, we evaluate on question answering tasks on different domains, specifically, the TREC COVID [56] and NFCorpus datasets [7] for the biomedical domain and FiQA [44] for the financial domain. We also evaluate on DBPedia [23] as an entity retrieval dataset and on Touché [6] as an argument retrieval dataset. We use the BEIR [63] versions of these datasets from the ir_datasets library.[3]

To explore the full capabilities of Hypencoder we want to evaluate how it performs on retrieval tasks that are more challenging than standard question-passage retrieval tasks. To some extent hardness is subjective, but we tried to define a clear set of criteria to define difficulty: (1) current neural retrieval models should struggle on the task, (2) term matching models like BM25 should also struggle on the task, (3) the queries are longer or otherwise more complicated than standard web queries. An additional requirement we had was that for tasks that were significantly different from the MSMARCO training data we wanted adequate training data to fine-tune the models before evaluation. We believe this is reasonable as we are not investigating the models' zero-shot performance but the inherent limits of the model.

The first dataset we select was the TREC Tip-of-the-Tongue (TOT) 2023 [3] that contains queries written by users that know many aspects of the item they are looking for but not the name of the item. Thus TOT queries tend to be verbose and can include many facets. The TREC TOT 2023 dataset specifically looks at TOT queries for movies with the corresponding movie's Wikipedia page as the golden passage. We use the development set as the test set relevance labels are not public yet. There are 150 queries. Each query has a single relevant passage. For training we use the data from Bhargav et al. [5] which is around 15k TOT queries from Reddit for the book and movie domain.

The second dataset is FollowIR [69] for instruction following in retrieval. This dataset is built on top of three existing TREC datasets: TREC Robust '04 [67], TREC News '21 [60], and TREC Core '17 [2]; it uses the fact that these datasets include instructions to the annotators which can act as a complex instruction. To test how well a retrieval system follows the instruction the creators of FollowIR modify the instruction to be more specific and re-annotate the known relevant documents. As training data we use MSMARCO with Instructions, a recent modification of MSMARCO which adds instructions to the queries as well as new positive passages and hard negative passages which consider the instruction [70].

The final dataset is a subset of TREC DL-HARD [43]. The full dataset uses some of the queries from TREC DL 2019 and 2020 as well as some queries that were considered for DL 2019 and 2020 but were not included in the final query collection. TREC DL-HARD is built specifically with the hardest queries from the TREC DL pool. The authors do so by using a commercial search engine to find queries that are not easily answered. The standard TREC DL-HARD dataset has 50 queries half of which appear in TREC DL 2019 or TREC DL 2020 and half of which are new queries which

[2]Available at https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-12-v2.

[3]Available at https://ir-datasets.com/.

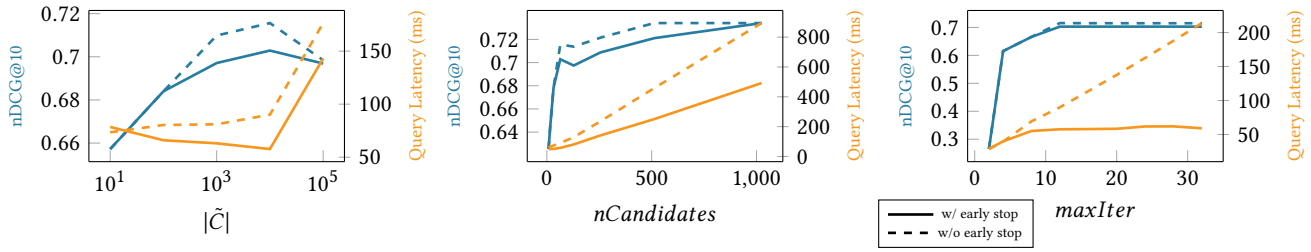

Figure 2: Relationship between the three main parameters of our efficient search: the size of the initial set of candidates $\tilde{C}$, the number of neighbors to explore *nCanddidates*, and the number of iterations *maxIter* and both effectiveness in terms of TREC DL '19 nDCG@10 and efficiency in terms of Query Latency.

are labeled by the authors of TREC DL-HARD. We found that the queries labeled by the authors had far fewer judged documents in the top 10 documents compared to those labeled by TREC (around 15% versus 93%), this made the evaluation metrics unreliable so we decided to only use those with TREC labeling.

4.2 Experimental Setup

4.2.1 Training Details. All the Hypencoders use BERT [16] base uncased as the base model. We use PyTorch and Huggingface for model implementation and training. All of our q-nets use a input dimension of 768 and hidden dimension of 768. Unless otherwise stated, we use 6 linear layers in the model not including the final output projection layer.

We use a training batch size of 64 per device and 128 in total. A single example in the batch is a query, positive document, and 8 additional documents ranging in relevance. The positive document is the top ranked document by our teacher model. The other 8 documents are sampled randomly from the passages associated with the query. For more details about the dataset see Section 4.1.1. Passages were truncated to 196 tokens and queries to 32 tokens.

Our primary loss function is Margin MSE [26]. When computing the loss, we construct (query, positive document, negative document) triplets where all of the negatives for a query form their own triplet. The loss is found by averaging the individual loss of all the triplets. In addition to Margin MSE, we use an in-batch cross entropy loss where the (query, positive document) is assumed to be a true positive and all the other queries' positive documents are assumed to be negatives. We do not consider the additional "hard" negatives from the query in the cross entropy loss as many of these documents are relevant to the query. We use AdamW as our optimizer with default settings and a learning rate of 2e-5 with a constant scheduler after a warm up of 6k steps. Our training hardware consist of two A100 GPUs with 80GB memory. Training took around 6 days.

To select the best model we evaluate each model on the validation set every 50k steps and pick the model with the best R Precision within the first 800k steps. We selected R Precision due to the fact that it balances both recall and precision in a single metric and does not require a predefined cutoff.

When training for the harder tasks we use AdamW with the learning rate 8e-6 with a linear scheduler and a warm-up ratio of 1e-1. For TOT training we train for 25 epochs or 3.3k steps. For FollowIR training we train for 1 epoch or around 10k steps. We use a batch-size of 96 and cross entropy loss. Each example in the batch includes a query, positive document, and hard negative document. We use a maximum document and query length of 512 tokens.

4.3 Baselines

For comparison with Hypencoder, we include several baseline models and models which we include for reference which are not directly comparable. Our main baselines which we evaluate on all datasets are TAS-B [28], CL-DRD [76], BM25, and our own bi-encoder baseline which we call BE-Base. We train BE-Base exactly the same as Hypencoder except we use separate encoders and use a linear LR scheduler. We select our main dense baselines TAS-B and CL-DRD as they are both strong bi-encoder models which leverage knowledge-distillation training and which use the same document embedding dimension of 768. For in-domain results we include an additional set of dense models: ANCE [73], TCT-ColBERT [37], and MarginMSE [26]. We only include these models in the in-domain results to save space in other sections and because TAS-B and CL-DRD outperform the other baselines. For reference we also include: the late-interaction model ColBERT v2 [58]; the neural sparse model SPLADE++ SD [17]; RepLLaMA a 7b parameter bi-encoder model [41]; DRAGON a bi-encoder trained with 5 teacher models and 20M synthetic queries; MonoBERT a reranking model reranking the top 1k BM25 retrieavls [50]; and the reranking model cross-SimLM reranking the top 200 passages from bi-SimLM [68]. Reference results were taken from the RepLLaMA [41] and DRAGON [38] papers.

4.4 Results and Discussion

4.4.1 In-Domain Results. Our in-domain results are presented in Table 1; they demonstrate that compared with baselines and even the reference models Hypencoder has very strong performance. Hypencoder is significantly better than each baseline in nDCG@10 on the combined TREC DL '19 and '20 and statistically better than all but CL-DRD on MSMARCO Dev RR@10. The most direct comparison, BE-Base, has far lower nDCG@10, RR, and RR@10 values indicating the Hypencoder is able to bring a large boost in precision based metrics over dense retrieval. In terms of recall Hypencoder is either as good or better than all the baselines though the gap is not as large as for precision based metrics.

Table 1: Comparison on in-domain evaluation datasets. The symbols next to each baseline indicate significance values with $p < 0.05$. Note, that † is a group of baselines.

	TREC-DL '19 & '20			MSMARCO Dev	
Model	nDCG@10	RR	R@1000	RR@10	R@1000
Single Vector Dense Retrieval Models & BM25 (Baselines)					
BM25 †	0.491	0.679	0.735	0.184	0.853
ANCE †	0.646	0.811	0.767	0.330	0.958
TCT-ColBERT †	0.669	0.820	0.806	0.335	0.964
Margin MSE †	0.669	0.845	0.782	0.325	0.955
TAS-B ♠	0.700	0.863	0.861	0.344	0.978
CL-DRD ♢	0.701	0.844	0.838	0.382	**0.981**
BE-Base ♣	0.713	0.855	0.868	0.359	0.980
Hypencoder	**0.736**$^{\dagger\spadesuit\diamond\clubsuit}$	**0.885**$^{\dagger\diamond}$	**0.871**$^{\dagger\diamond}$	**0.386**$^{\dagger\spadesuit\clubsuit}$	**0.981**$^{\dagger\spadesuit}$
Other Retrieval Models (Reference Models)					
ColBERT v2	0.749	-	-	0.397	0.984
SPLADE++ SD	0.723	-	-	0.368	0.979
RepLLaMA	0.731	-	-	0.412	0.994
DRAGON	0.734	-	-	0.393	0.985
MonoBERT	0.722	-	-	0.372	0.853
cross-SimLM	0.735	-	-	0.437	0.987

Impressively Hypencoder is able to surpass DRAGON on nDCG@10 on the combined TREC DL '19 and '20 query set, though DRAGON uses the same base model and is a bi-encoder, it uses 32 A100s to train, 40x the training queries, and a complex 5 teacher curriculum learning distillation training technique. In other words, DRAGON is likely close to if not the ceiling for BERT-based bi-encoders and still Hypencoder is able to match it with a simple distillation training setup and far less training compute.

Hypencoder also beats both rerankers MonoBERT and cross-SimLM; demonstrating that reranking cannot make up for a weak retriever's performance. Continuing in TREC-DL '19 and '20 we find that Hypencoder even surpassed RepLLaMA which is more than 60x larger and which also uses a significantly larger document embedding dimension of 4096. In fact the only model beating Hypencoder in nDCG@10 is ColBERTv2 which uses an embedding for every token in the document compared to Hypencoder's fixed 768 dimension token. MSMARCO Dev results are also good with Hypencoder outperforming all the baselines and outperforming a few of the reference models such as SPLADE++ and MonoBERT.

Overall Hypencoder's in-domain results are exceptionally strong given the simple training routine used, small encoder model size, and document representation size. To the best of our knowledge, Hypencoder sets a new record for combined TREC-DL '19 and '20 nDCG@10 with a 768 dimension dense document vector.

4.4.2 Out-of-Domain Results. Table 2 shows our results on the select out-of-domain datasets, we only include our main baseline models and BM25 due to space limitations. The general trend is that Hypencoder has strong out-of-domain performance in question answering tasks (Q&A) and entity retrieval tasks. This indicates that despite Hypencoder's more complex similarity function it is still able to generalize well in a zero-shot manner to new tasks.

4.4.3 Results on Harder Retrieval Tasks. The results on the harder retrieval tasks are in Table 3, like in the out-of-domain section we only consider the main baseline models and BM25. We can see that in the harder tasks Hypencoder remains dominant over

Table 2: Out-of-domain results in nDCG@10. We only compare significance with BE-Base. Significance results with $p < 0.05$ are shown with the ♣ and $p < 0.1$ are shown with ♢.

	Baselines				Ours
Rep type	sparse	dense	dense	dense	hypernet
	BM25	TAS-B	CL-DRD	BE-Base	Hypecoder
Q & A					
TREC-Covid	0.656	0.481	0.584	0.651	**0.688**$^{\diamond}$
FiQA	0.236	0.300	0.308	0.309	**0.314**
NFCorpus	**0.325**	0.319	0.315	0.327	0.324
Misc.					
DBPedia	0.313	0.384	0.381	0.405	**0.419**$^{\clubsuit}$
Touché v2	**0.367**	0.162	0.203	0.240	0.258$^{\diamond}$

the baseline models with higher retrieval metrics in all but one column. Additionally, the relative improvement compared to the in-domain results are higher (on all metrics that Hypencoder is the best for) suggesting that on harder tasks the added complexity that can be captured by Hypencoder's similarity function is especially important. Additionally the high performance on TREC tip-of-the-tongue (TOT) and FollowIR indicate that Hypencoder adapts well to different domains through domain-specific fine-tuning.

On the evaluated subset of TREC DL-HARD we see that Hypencoder has stronger precision metrics than the baselines by a large margin. As mentioned previously the higher relative improvement suggests that Hypencoder is especially dominant on harder tasks which, in part, explains its higher performance on TREC DL '19 and '20. Though on in-domain dataset Hypencoder does better or the same on recall metrics, on TREC DL-HARD BE-Base has higher recall than Hypencoder. We suspect that this may be because the relevance function that the q-net applies is not smooth, which has the benefit of being more discerning and likely accounts for some of the precision gains. However, if the q-net makes a mistake the non-smooth scoring could result in a much harsher score than the linear inner product is capable of producing.

Moving to TREC tip-of-my-tongue (TOT) we see that Hypencoder continues to perform well. Tip-of-the-tongue is a complex retrieval task with long queries and passages and multiple aspects, the fact Hypencoder outperforms the baselines by a large margin validates the need for a more complex relevance function.

Finally we have FollowIR which has three subsets – on all three Hypencoder has the best performance on the retrieval evaluation metrics of choice, in many cases by a sizable amount. Beside the retrieval evaluation metrics we also include p-MRR which is a metric released in the FollowIR [69] paper. The metric measures the change in document ranks before and after an instruction is modified to see how well the model responses to the additional requirements. A p-MRR of 0 indicates no change in document rank based on the instruction change and a p-MRR of +100 indicates the documents were perfectly changed based on the instruction while -100 indicates the opposite. For additional details we refer readers to the original FollowIR paper [69]. As p-MRR is relative to each model's performance before the instructions are modified it is not indicative of stand-alone retrieval performance. With that said, Hypencoder is the only model to achieve a positive p-MRR

Table 3: Evaluation metrics for the harder set of tasks which include TREC DL-HARD, TREC Tip-of-my-tongue TOT, and FollowIR. Significance is shown at $p < 0.1$. For FollowIR we do not perform significance tests on BM25.

	TREC DL-HARD			TREC TOT DEV			FollowIR Robust '04		FollowIR News '21		FollowIR Core '17	
Model	nDCG@10	RR	R@1000	nDCG@10	RR	nDCG@1000	AP	p-MRR	nDCG@5	p-MRR	AP	p-MRR
BM25 †	0.466	0.813	0.646	0.086	0.088	0.131	0.121	**-3.1**	0.193	-2.1	0.081	**-1.1**
TAS-B ♠	0.574	0.789	0.777	0.097	0.089	0.162	0.203	-5.4	0.263	-0.8	0.170	-10.0
CL-DRD ♢	0.573	0.790	0.719	0.088	0.082	0.151	0.206	-7.2	0.240	-0.3	0.162	-12.1
BE-Base ♣	0.607	0.864	**0.805**	0.121	0.110	0.179	0.207	-3.7	0.239	-1.1	0.178	-7.7
Hypencoder	**0.630**$^{\dagger\spadesuit\diamond}$	**0.887**$^{\spadesuit\diamond}$	0.798$^{\dagger\diamond}$	**0.134**$^{\dagger\spadesuit\diamond}$	**0.125**$^{\spadesuit\diamond}$	**0.182**$^{\diamond\dagger}$	**0.212**$^{\spadesuit}$	-3.5	**0.272**	**2.0**	**0.193**	-11.8

indicating it correctly modified the document ranking based on the instruction. This is no small feat as in the original FollowIR paper no retrieval model of the size of Hypencoder was able to get a positive p-MRR and even many much larger models trained on large instruction retrieval datasets could not get a positive result.

4.4.4 Analysis of Efficiency. To be able to search large-scale collections Hypencoder has to work well while only doing computation on a small subset of the document corpus. This led us to develop the efficient algorithm in Section 3.6. To quantify the performance we do a set of experiments varying the key parameters to see how each impacts both search quality and query latency. The results of these experiments can be seen in Figure 2. Note the document-to-document graph used has 100 neighbors per document. The figures show that each parameter has an important role in search quality. The largest role is played by *maxIter* which can drastically reduce search performance if not set high enough, but which plateaus after around 12. The value of *nCandidates* follows a similar pattern with a drastic increase followed by a plateau. The parameter $\tilde{C}$, the number of initial candidates, is unique in that it has a more gradual rise and is the only parameter that causes a decrease in effectiveness if raised too high. We suspect this happens because only *nCandidates* are explored at each iteration but all candidates in $\tilde{C}$ are considered visited and thus are not explored in the future. This could mean that many good results from the initial candidates are not explored, leaving potentially good neighbors of these nodes undiscovered. Another interesting aspect of $\tilde{C}$ is that time decreases as it increases when early stopping is used. This is likely because high quality candidates are found sooner allowing search to end more quickly. In general, our approximation seems to behave as expected in terms of both time and retrieval performance.

With the insights from our analysis above we developed two configurations, one which optimizes for speed and one that optimizes for retrieval quality. These two configurations compared against exhaustive search can be seen in Table 4. Both approximate configurations significantly decrease the retrieval time when compared to the exhaustive approach.

4.4.5 Impact of q-net Depth. Hypencoder performance suggests that having a more complex relevance function does indeed help improve retrieval performance as we had hypothesized. This raises the question: what is the optimal complexity of this relevance function. To answer this question we trained four versions of Hypencoder each trained to produce a q-net with a different number of layers. We selected *[2, 4, 6, 8]* as the number of q-net layers.

The results can be seen in Figure 3. The experiment shows that there is a benefit beyond two layers and that at least four is required for the best performance. Performance stays the same with six, indicating that this might be the point of diminishing returns. Lastly, eight layers decrease performance. There could be a number of reasons for this including that eight layers are harder to optimize or because effectively learning how to use all eight layers takes longer and thus might achieve better performance with extended training time.

Table 4: Average query latency and nDCG@10 on TREC DL '19 and '20 with efficient search. Efficient 1 uses parameters ($\tilde{C}$ = 10000, *nCandidates* = 64, *maxIter* = 16), Efficient 2 uses parameters ($\tilde{C}$ = 100000, *nCandidates* = 328, *maxIter* = 20). All model inference was performed on an NVIDIA L40S with BF16 precision.

Search Type	Query Lat. (ms)	DL '19	DL '20
Exhaustive	1769.8	0.742	0.731
Efficient 1	59.6	0.702	0.730
Efficient 2	231.1	0.722	0.731

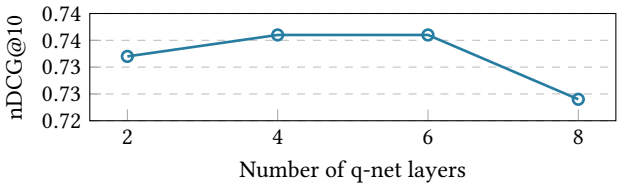

Figure 3: Average nDCG@10 on TREC DL '19 and '20 versus number of layers in the q-net.

5 CONCLUSION

We propose a new class of retrieval model, the Hypencoder which overcomes the limitations of inner product based similarity functions that we prove to exist. Our model achieves a new state-of-the-art on TREC DL '19 and '20 for BERT sized encoder models with a single dense document vector and shows even stronger relative improvement on harder retrieval tasks such as tip-of-the-tongue queries. Further we demonstrate that learned relevance models can be applied to large-scale search corpus in an efficient way with our proposed approximate search algorithm. As Hypencoder is a flexible framework there is much interesting future work to explore, such as multi-vector document representations and corpus pretraining to name a few.

ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval, in part by the NSF Graduate Research Fellowships Program (GRFP) Award #1938059, and in part by the Office of Naval Research contract number N000142412612. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

REFERENCES

[1] Abien Fred Agarap. 2018. Deep Learning using Rectified Linear Units (ReLU). *CoRR* abs/1803.08375 (2018). arXiv:1803.08375 http://arxiv.org/abs/1803.08375

[2] James Allan, Donna K. Harman, E. Kanoulas, Dan Li, Christophe Van Gysel, and Ellen M. Voorhees. 2017. TREC 2017 Common Core Track Overview. In *Text Retrieval Conference*. https://api.semanticscholar.org/CorpusID:38019792

[3] Jaime Arguello, Samarth Bhargav, Fernando Diaz, Evangelos Kanoulas, and Bhaskar Mitra. 2023. Overview of the TREC 2023 Tip-of-the-Tongue Track. In *The Thirty-Second Text REtrieval Conference Proceedings (TREC 2023), Gaithersburg, MD, USA, November 14-17, 2023 (NIST Special Publication, Vol. 500-xxx)*, Ian Soboroff and Angela Ellis (Eds.). National Institute of Standards and Technology (NIST). https://trec.nist.gov/pubs/trec32/papers/Overview_tot.pdf

[4] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR* abs/1607.06450 (2016). arXiv:1607.06450 http://arxiv.org/abs/1607.06450

[5] Samarth Bhargav, Georgios Sidiropoulos, and Evangelos Kanoulas. 2022. 'It's on the tip of my tongue': A new Dataset for Known-Item Retrieval. In *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining, Virtual Event / Tempe, AZ, USA, February 21 - 25, 2022*, K. Selcuk Candan, Huan Liu, Leman Akoglu, Xin Luna Dong, and Jiliang Tang (Eds.). ACM, 48–56. https://doi.org/10.1145/3488560.3498421

[6] Alexander Bondarenko, Maik Fröbe, Meriem Beloucif, Lukas Gienapp, Yamen Ajjour, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. 2020. Overview of Touché 2020: Argument Retrieval. In *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020 (CEUR Workshop Proceedings, Vol. 2696)*, Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol (Eds.). CEUR-WS.org. https://ceur-ws.org/Vol-2696/paper_261.pdf

[7] Vera Boteva, Demian Gholipour Ghalandari, Artem Sokolov, and Stefan Riezler. 2016. A Full-Text Learning to Rank Dataset for Medical Information Retrieval. In *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings (Lecture Notes in Computer Science, Vol. 9626)*, Nicola Ferro, Fabio Crestani, Marie-Francine Moens, Josiane Mothe, Fabrizio Silvestri, Giorgio Maria Di Nunzio, Claudia Hauff, and Gianmaria Silvello (Eds.). Springer, 716–722. https://doi.org/10.1007/978-3-319-30671-1_58

[8] Christopher Burges, Robert Ragno, and Quoc Le. 2006. Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems* 19 (2006).

[9] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. 2005. Learning to rank using gradient descent. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005 (ACM International Conference Proceeding Series, Vol. 119)*, Luc De Raedt and Stefan Wrobel (Eds.). ACM, 89–96. https://doi.org/10.1145/1102351.1102363

[10] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2020. Overview of the TREC 2020 Deep Learning Track. In *Proceedings of the Twenty-Ninth Text REtrieval Conference, TREC 2020, Virtual Event [Gaithersburg, Maryland, USA], November 16-20, 2020 (NIST Special Publication, Vol. 1266)*, Ellen M. Voorhees and Angela Ellis (Eds.). National Institute of Standards and Technology (NIST). https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.DL.pdf

[11] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Jimmy Lin. 2021. Overview of the TREC 2021 Deep Learning Track. In *Proceedings of the Thirtieth Text REtrieval Conference, TREC 2021, online, November 15-19, 2021 (NIST Special Publication, Vol. 500-335)*, Ian Soboroff and Angela Ellis (Eds.). National Institute of Standards and Technology (NIST). https://trec.nist.gov/pubs/trec30/papers/Overview-DL.pdf

[12] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, Jimmy Lin, Ellen M. Voorhees, and Ian Soboroff. 2022. Overview of the TREC 2022 Deep Learning Track. In *Proceedings of the Thirty-First Text REtrieval Conference, TREC 2022, online, November 15-19, 2022 (NIST Special Publication, Vol. 500-338)*, Ian Soboroff and Angela Ellis (Eds.). National Institute of Standards and Technology (NIST). https://trec.nist.gov/pubs/trec31/papers/Overview_deep.pdf

[13] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2020. Overview of the TREC 2019 deep learning track. *CoRR* abs/2003.07820 (2020). arXiv:2003.07820 https://arxiv.org/abs/2003.07820

[14] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. 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, 65–74. https://doi.org/10.1145/3077136.3080832

[15] Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. 2024. Language Modeling Is Compression. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=jznbgiynus

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/V1/N19-1423

[17] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, Enrique Amigó, Pablo Castells, Julio Gonzalo, Ben Carterette, J. Shane Culpepper, and Gabriella Kazai (Eds.). ACM, 2353–2359. https://doi.org/10.1145/3477495.3531857

[18] 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*. Association for Computing Machinery, New York, NY, USA, 2288–2292. https://doi.org/10.1145/3404835.3463098

[19] Luyu Gao and Jamie Callan. 2022. Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 2843–2853. https://doi.org/10.18653/V1/2022.ACL-LONG.203

[20] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List. In *North American Chapter of the Association for Computational Linguistics*. https://api.semanticscholar.org/CorpusID:233241070

[21] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W. Bruce Croft, and Xueqi Cheng. 2020. A Deep Look into neural ranking models for information retrieval. *Information Processing & Management* 57, 6 (2020), 102067. https://doi.org/10.1016/j.ipm.2019.102067

[22] David Ha, Andrew M. Dai, and Quoc V. Le. 2017. HyperNetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rkpACe1lx

[23] Faegheh Hasibi, Fedor Nikolaev, Chenyan Xiong, Krisztian Balog, Svein Erik Bratsberg, Alexander Kotov, and Jamie Callan. 2017. DBpedia-Entity v2: A Test Collection for Entity Search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White (Eds.). ACM, 1265–1268. https://doi.org/10.1145/3077136.3080751

[24] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White (Eds.). ACM, 355–364. https://doi.org/10.1145/3077136.3080777

[25] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich (Eds.). ACM, 173–182. https://doi.org/10.1145/3038912.3052569

[26] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020. Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation. *CoRR* abs/2010.02666 (2020). arXiv:2010.02666 https://arxiv.org/abs/2010.02666

[27] Sebastian Hofstätter, O. Khattab, Sophia Althammer, Mete Sertkan, and Allan Hanbury. 2022. colberter. *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (2022). https://api.semanticscholar.org/CorpusID:247628023

[28] Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. 2021. Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling. In *SIGIR '21: The 44th International ACM SIGIR Conference*

on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 113–122. https://doi.org/10.1145/3404835.3462891

[29] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5 (1989), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

[30] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Trans. Mach. Learn. Res.* 2022 (2022). https://openreview.net/forum?id=jKN1pXi7b0

[31] Ziwei Ji, Himanshu Jain, Andreas Veit, Sashank J. Reddi, Sadeep Jayasumana, Ankit Singh Rawat, Aditya Krishna Menon, Felix Yu, and Sanjiv Kumar. 2024. Efficient Document Ranking with Learnable Late Interactions. *CoRR* abs/2406.17968 (2024). https://doi.org/10.48550/ARXIV.2406.17968 arXiv:2406.17968

[32] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6769–6781. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.550

[33] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, Jimmy X. Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (Eds.). ACM, 39–48. https://doi.org/10.1145/3397271.3401075

[34] Minghan Li, Sheng-Chieh Lin, Barlas Oğuz, Asish Ghoshal, Jimmy J. Lin, Yashar Mehdad, Wen tau Yih, and Xilun Chen. 2022. CITADEL: Conditional Token Interaction via Dynamic Lexical Routing for Efficient and Effective Multi-Vector Retrieval. In *Annual Meeting of the Association for Computational Linguistics*. https://api.semanticscholar.org/CorpusID:253708231

[35] Yury Lifshits and Shengyu Zhang. 2009. Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, Claire Mathieu (Ed.). SIAM, 318–326. https://doi.org/10.1137/1.9781611973068.36

[36] J. Lin, R. Nogueira, and A. Yates. 2022. *Pretrained Transformers for Text Ranking: BERT and Beyond.* Springer International Publishing.

[37] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2020. Distilling Dense Representations for Ranking using Tightly-Coupled Teachers. *CoRR* abs/2010.11386 (2020). arXiv:2010.11386 https://arxiv.org/abs/2010.11386

[38] Sheng-Chieh Lin, Akari Asai, Minghan Li, Barlas Oguz, Jimmy Lin, Yashar Mehdad, Wen-tau Yih, and Xilun Chen. 2023. How to Train Your Dragon: Diverse Augmentation Towards Generalizable Dense Retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 6385–6400. https://doi.org/10.18653/v1/2023.findings-emnlp.423

[39] Zhenghao Lin, Yeyun Gong, Xiao Liu, Hang Zhang, Chen Lin, Anlei Dong, Jian Jiao, Jingwen Lu, Daxin Jiang, Rangan Majumder, and Nan Duan. 2023. PROD: Progressive Distillation for Dense Retrieval. In *Proceedings of the ACM Web Conference 2023* (Austin, TX, USA) *(WWW '23)*. Association for Computing Machinery, New York, NY, USA, 3299–3308. https://doi.org/10.1145/3543507.3583421

[40] Xinyu Ma, Jiafeng Guo, Ruqing Zhang, Yixing Fan, Yingyan Li, and Xueqi Cheng. 2021. B-PROP: Bootstrapped Pre-training with Representative Words Prediction for Ad-hoc Retrieval. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2021). https://api.semanticscholar.org/CorpusID:233307194

[41] 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, SIGIR 2024, Washington DC, USA, July 14-18, 2024*, Grace Hui Yang, Hongning Wang, Sam Han, Claudia Hauff, Guido Zuccon, and Yi Zhang (Eds.). ACM, 2421–2425. https://doi.org/10.1145/3626772.3657951

[42] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. 2020. Efficient Document Re-Ranking for Transformers by Precomputing Term Representations. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, Jimmy X. Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (Eds.). ACM, 49–58. https://doi.org/10.1145/3397271.3401093

[43] Iain Mackie, Jeffrey Dalton, and Andrew Yates. 2021. How Deep is your Learning: the DL-HARD Annotated Deep Learning Dataset. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval.*

[44] Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. 2018. WWW'18 Open Challenge: Financial Opinion Mining and Question Answering. In *Companion of the The Web Conference 2018 on The Web Conference 2018, WWW 2018, Lyon , France, April 23-27, 2018*, Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 1941–1942. https://doi.org/10.1145/3184558.3192301

[45] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[46] Bhaskar Mitra and Nick Craswell. 2018. An Introduction to Neural Information Retrieval. *Found. Trends Inf. Retr.* 13, 1 (Dec. 2018), 1–126. https://doi.org/10.1561/1500000061

[47] Bhaskar Mitra and Nick Craswell. 2019. An Updated Duet Model for Passage Re-ranking. *CoRR* abs/1903.07666 (2019). arXiv:1903.07666 http://arxiv.org/abs/1903.07666

[48] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to Match using Local and Distributed Representations of Text for Web Search. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich (Eds.). ACM, 1291–1299. https://doi.org/10.1145/3038912.3052579

[49] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016 (CEUR Workshop Proceedings, Vol. 1773)*, Tarek Richard Besold, Antoine Bordes, Artur S. d'Avila Garcez, and Greg Wayne (Eds.). CEUR-WS.org. https://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper9.pdf

[50] Rodrigo Frassetto Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-Stage Document Ranking with BERT. *CoRR* abs/1910.14424 (2019). arXiv:1910.14424 http://arxiv.org/abs/1910.14424

[51] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text Matching as Image Recognition. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, Dale Schuurmans and Michael P. Wellman (Eds.). AAAI Press, 2793–2799. https://doi.org/10.1609/AAAI.V30I1.10341

[52] Prafull Prakash, Julian Killingback, and Hamed Zamani. 2021. Learning Robust Dense Retrieval Models from Incomplete Relevance Labels. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 1728–1732. https://doi.org/10.1145/3404835.3463106

[53] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 5835–5847. https://doi.org/10.18653/V1/2021.NAACL-MAIN.466

[54] Johann Radon. 1921. Mengen konvexer Körper, die einen gemeinsamen Punkt enthalten. *Math. Ann.* 83, 1 (1921). https://doi.org/10.1007/BF01464231

[55] Nils Reimers and Iryna Gurevych. 2019. 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. https://arxiv.org/abs/1908.10084

[56] Kirk Roberts, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, Kyle Lo, Ian Soboroff, Ellen M. Voorhees, Lucy Lu Wang, and William R. Hersh. 2021. Searching for scientific evidence in a pandemic: An overview of TREC-COVID. *J. Biomed. Informatics* 121 (2021), 103865. https://doi.org/10.1016/J.JBI.2021.103865

[57] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. 2019. Meta-Learning with Latent Embedding Optimization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=BJgklhAcK7

[58] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (Eds.). Association for Computational Linguistics, Seattle, United States, 3715–3734. https://doi.org/10.18653/v1/2022.naacl-main.272

[59] Marcin Sendera, Marcin Przewiezlikowski, Konrad Karanowski, Maciej Zieba, Jacek Tabor, and Przemyslaw Spurek. 2023. HyperShot: Few-Shot Learning by Kernel HyperNetworks. In *IEEE/CVF Winter Conference on Applications of*

Computer Vision, WACV 2023, Waikoloa, HI, USA, January 2-7, 2023. IEEE, 2468–2477. https://doi.org/10.1109/WACV56688.2023.00250

[60] Ian Soboroff, Shudong Huang, and Donna Harman. 2020. TREC 2020 News Track Overview. In *Proceedings of the Twenty-Ninth Text REtrieval Conference, TREC 2020, Virtual Event [Gaithersburg, Maryland, USA], November 16-20, 2020 (NIST Special Publication, Vol. 1266)*, Ellen M. Voorhees and Angela Ellis (Eds.). National Institute of Standards and Technology (NIST). https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.N.pdf

[61] Shulong Tan, Weijie Zhao, and Ping Li. 2021. Fast Neural Ranking on Bipartite Graph Indices. *Proc. VLDB Endow.* 15, 4 (2021), 794–803. https://doi.org/10.14778/3503585.3503589

[62] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2020. Fast Item Ranking under Neural Network based Measures. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang (Eds.). ACM, 591–599. https://doi.org/10.1145/3336191.3371830

[63] 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 *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, Joaquin Vanschoren and Sai-Kit Yeung (Eds.). https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html

[64] V. N. Vapnik and A. Ya. Chervonenkis. 1971. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability & Its Applications* 16, 2 (1971), 264–280. https://doi.org/10.1137/1116025

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[66] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. 2020. Continual learning with hypernetworks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=SJgwNerKvB

[67] Ellen M. Voorhees. 2004. Overview of the TREC 2004 Robust Track. In *Proceedings of the Thirteenth Text REtrieval Conference, TREC 2004, Gaithersburg, Maryland, USA, November 16-19, 2004 (NIST Special Publication, Vol. 500-261)*, Ellen M. Voorhees and Lori P. Buckland (Eds.). National Institute of Standards and Technology (NIST). http://trec.nist.gov/pubs/trec13/papers/ROBUST.OVERVIEW.pdf

[68] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2023. SimLM: Pre-training with Representation Bottleneck for Dense Passage Retrieval. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 2244–2258. https://doi.org/10.18653/v1/2023.acl-long.125

[69] Orion Weller, Benjamin Chang, Sean MacAvaney, Kyle Lo, Arman Cohan, Benjamin Van Durme, Dawn J. Lawrie, and Luca Soldaini. 2024. FollowIR: Evaluating and Teaching Information Retrieval Models to Follow Instructions. *CoRR* abs/2403.15246 (2024). https://doi.org/10.48550/ARXIV.2403.15246 arXiv:2403.15246

[70] Orion Weller, Benjamin Van Durme, Dawn J. Lawrie, Ashwin Paranjape, Yuhao Zhang, and Jack Hessel. 2024. Promptriever: Instruction-Trained Retrievers Can Be Prompted Like Language Models. *CoRR* abs/2409.11136 (2024). https://doi.org/10.48550/ARXIV.2409.11136 arXiv:2409.11136

[71] Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. 2022. RetroMAE: Pre-Training Retrieval-oriented Language Models Via Masked Auto-Encoder. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, 538–548. https://doi.org/10.18653/V1/2022.EMNLP-MAIN.35

[72] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White (Eds.). ACM, 55–64. https://doi.org/10.1145/3077136.3080809

[73] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=zeFrfgyZln

[74] Zhou Yang, Qingfeng Lan, Jiafeng Guo, Yixing Fan, Xiaofei Zhu, Yanyan Lan, Yue Wang, and Xueqi Cheng. 2018. A Deep Top-K Relevance Matching Model for Ad-hoc Retrieval. In *Information Retrieval - 24th China Conference, CCIR 2018, Guilin, China, September 27-29, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11168)*, Shichao Zhang, Tie-Yan Liu, Xianxian Li, Jiafeng Guo, and Chenliang Li (Eds.). Springer, 16–27. https://doi.org/10.1007/978-3-030-01012-6_2

[75] Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik G. Learned-Miller, and Jaap Kamps. 2018. 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, CIKM 2018, Torino, Italy, October 22-26, 2018*, Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang (Eds.). ACM, 497–506. https://doi.org/10.1145/3269206.3271800

[76] Hansi Zeng, Hamed Zamani, and Vishwa Vinay. 2022. Curriculum Learning for Dense Retrieval Distillation. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, Enrique Amigó, Pablo Castells, Julio Gonzalo, Ben Carterette, J. Shane Culpepper, and Gabriella Kazai (Eds.). ACM, 1979–1983. https://doi.org/10.1145/3477495.3531791

[77] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, M. Zhang, and Shaoping Ma. 2021. Optimizing Dense Retrieval Model Training with Hard Negatives. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2021). https://api.semanticscholar.org/CorpusID:233289894

[78] Chris Zhang, Mengye Ren, and Raquel Urtasun. 2019. Graph HyperNetworks for Neural Architecture Search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=rkgW0oA9FX

[79] Weijie Zhao, Shulong Tan, and Ping Li. 2024. GUITAR: Gradient Pruning toward Fast Neural Ranking. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2024, Washington DC, USA, July 14-18, 2024*, Grace Hui Yang, Hongning Wang, Sam Han, Claudia Hauff, Guido Zuccon, and Yi Zhang (Eds.). ACM, 163–173. https://doi.org/10.1145/3626772.3657728