Ultra-High Dimensional Sparse Representations with Binarization for Efficient Text Retrieval

Kyoung-Rok Jang[1], Junmo Kang[1][*], Giwon Hong[1][*], Sung-Hyon Myaeng[1][†],
Joohee Park[2], Taewon Yoon[2], Heecheol Seo[2]
[1]KAIST, Republic of Korea
[2]NAVER, Republic of Korea
{kyoungrok.jang, junmo.kang, giwon.hong, myaeng}@kaist.ac.kr
{james.joohee.park, taewon.yoon, heecheol.seo}@navercorp.com

Abstract

The semantic matching capabilities of neural information retrieval can ameliorate synonymy and polysemy problems of symbolic approaches. However, neural models' dense representations are more suitable for re-ranking, due to their inefficiency. Sparse representations, either in symbolic or latent form, are more efficient with an inverted index. Taking the merits of the sparse and dense representations, we propose an ultra-high dimensional (UHD) representation scheme equipped with directly controllable sparsity. UHD's large capacity and minimal noise and interference among the dimensions allow for *binarized* representations, which are highly efficient for storage and search. Also proposed is a bucketing method, where the embeddings from multiple layers of BERT are selected/merged to represent diverse linguistic aspects. We test our models with MS MARCO and TREC CAR, showing that our models outperforms other sparse models.

1 Introduction

Using neural models for representing and processing textual data has become a de-facto standard. Recent approaches to information retrieval (IR) and natural language processing (NLP) tasks adopt contextual language models (e.g., BERT (Devlin et al., 2019)), ameliorating both synonymy and polysemy problems associated with words (Zhan et al., 2020; Khattab and Zaharia, 2020; Qu et al., 2021). In these approaches, queries and documents are first encoded into contextual embeddings, either independently (Zhan et al., 2020) or with interactions (Qu et al., 2021), resulting in low-dimensional dense representations. Then the documents are retrieved based on a similarity metric, such as cosine similarity, defined for two vectors.

Despite the impressive results, the inherent properties of dense representations — low dimensional and dense — can pose a severe efficiency challenge for first-stage or full ranking. Since each dimension in a dense embedding is overloaded and entangled (i.e., polysemous) with the limited number of dimensions available, it is susceptible to false matches with large index sizes (Reimers and Gurevych, 2020). Also, all the dimensions must participate in representing words, queries, and documents regardless of the amount of information content as well as in matching (Zamani et al., 2018), which is inefficient. As a result, it is meaningless to build an inverted index for the dimensions, without which it is difficult to build an efficient and effective first-stage or full ranker.

Other drawbacks of the dense embedding approaches to IR include: 1) The retrieval results are hardly interpretable like other neural approaches, making it difficult to improve the design through failure analyses or implement conditional/selective processing (Belinkov and Glass, 2019); and 2) It is not straightforward to adopt the well studied term-based symbolic IR techniques, such as pseudo-relevance feedback, for further improvements.

Our focus is to propose a full-blown neural first-stage ranker that alleviates the shortcomings of dense neural IR and yet achieves competitive effectiveness. The main thrust of our method is to utilize ultra-high dimensional (UHD) embeddings[1] with high sparsity, possessing superior expressive and discriminating power to the extent that they are binarized. In addition, our proposed approach has the potential to make individual dimensions of the document/query embeddings interpretable (Faruqui et al., 2015; Sun et al., 2016) and support mutually non-interfering aggregation of multiple embeddings (Ahmad and Hawkins, 2015).

In order to obtain UHD embeddings, we train the $Winner\text{-}Take\text{-}All$ (WTA) model (Ahmad and

---

[*]These authors contributed equally.
[†]The corresponding author

[1]The dimension size is close to half a million.

Hawkins, 2015; Makhzani and Frey, 2015) on top of BERT with a new learning objective for IR. WTA model is fundamentally a linear layer that only preserves top-k activation and sets the others to zero. WTA is chosen because we can precisely control the outputs' sparsity by explicitly setting the parameter k. In contrast, the method relying on regularization for sparsity (e.g., minimizing L1-norm) (Faruqui et al., 2015; Sun et al., 2016; Zamani et al., 2018) is neither straightforward nor precise in controlling the degree of sparsity.

With the large capacity and robustness against noise and interference, UHD allows for binarized representations so that matching can be further simplified with little degradation in effectiveness. This capability of UHD achieves extreme efficiency in terms of storage and speed, making it possible to build a stand-alone neural IR model for an industry.

Besides the efficiency goal, we also attempt to represent different aspects of linguistic properties of documents and queries. Instead of the common approach of using only the final layer of BERT, we make use of multiple layers, each of which emits token representations for different linguistic properties (Jo and Myaeng, 2020), by devising a bucketing mechanism.

We evaluate our model on MS MARCO passage retrieval (Bajaj et al., 2018) and TREC CAR (Dietz et al., 2017), showing that it outperforms previous sparse models and is competitive with dense models for effectiveness. Even though it is a neural model, our UHD-based IR method with binarization is highly efficient, on par with BM25, surpassing all the sparse models.

2 Related Work

State-of-the-art neural retrieval models (Nogueira and Cho, 2019; Qu et al., 2021) adopt a cross-encoder that shows high effectiveness while known to be impractical for large-scale retrieval (Luan et al., 2021). The cross-encoder requires that a query and documents must be encoded together, limiting models to a re-ranker. Dense neural models have been proposed for first-stage retrieval. Rep-BERT (Zhan et al., 2020) encodes queries and documents separately and ranks using an inner product. It achieves efficiency by relying heavily on GPUs. ColBERT (Khattab and Zaharia, 2020) positions itself between the cross-encoder and inner product by proposing a late interaction for scoring. Despite its promising performance, it remains questionable

whether first-stage retrieval is computationally feasible at a large scale (Bai et al., 2020), even with an external library Faiss (Johnson et al., 2019).

Sparse models are attractive for first-stage retrieval (Dai and Callan, 2020b,a; Nogueira et al., 2019; Nogueira, 2019; Bai et al., 2020). Their retrieval speed comes from the sparsity that enables to leverage an efficient inverted index. On the other hand, they usually lack the ability to use non-symbolic latent semantics that can be captured by neural models. SOLAR (Medini et al., 2020) proposes randomly initialized high-dimensional and ultra-sparse embeddings for book classification but ignores their content, making it unsuitable for IR tasks. SNRM (Zamani et al., 2018) creates a sparse latent representation using a fully connected layer similar to an autoencoder. It is limited with a simple word embedding, lower dimensionality (20K), uncontrollable sparsity.

3 Sparsified and Bucketized Embeddings

3.1 Design Objectives

Our goal is to build an efficient neural ranker without relying on external efficiency-enhancing measures like Approximate Nearest Neighbor (ANN) (Johnson et al., 2019; Jegou et al., 2010). We posit that it is crucial to represent the distinct textual signals in the embeddings for enhanced discriminative power, thereby achieving the effect of symbol-like characteristics with neural approaches.

Despite the contextualized language modeling capabilities of the recent transformer architectures, we note the shortcomings of the low-dimensional dense representations for IR: 1) All the dimensions must always be accessed during document-query matching; 2) Dimensions are highly overloaded and polysemous that they hardly serve as useful discriminators (e.g., Arora et al., 2018; Reimers and Gurevych, 2020); 3) Various linguistic properties are entangled or under-represented.

As a result, we attempt to enforce the following:

- The embeddings need to be sufficiently high dimensional and sparse, so that they can be processed efficiently for matching while enjoying the high expressive power.

- Each dimension represents a specific concept, making it suitable for precise semantic computation and more interpretable.

- Different aspects of queries and documents should be captured for versatile representa-
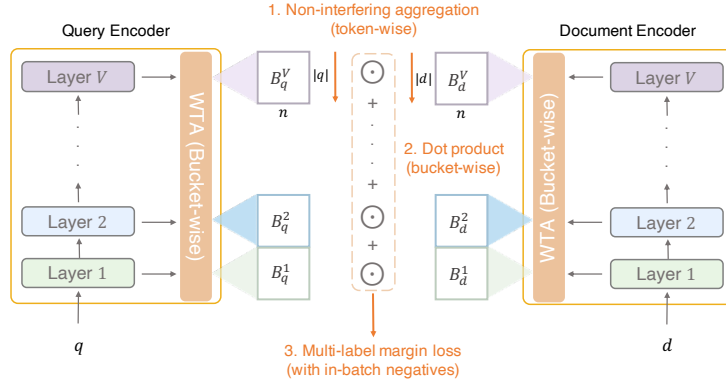
Figure 1: Overall architecture with the training scheme for building UHD sparse representations.

tions as IR queries are notoriously ambiguous with diverse intents (Azad and Deepak, 2019).

Aside from representation perspectives, we adhere to the following requirements for efficiency:

- The ranker must support offline encoding of documents to build an inverted index.

- The matching function should ignore a signal if it is not strong enough and avoid unnecessary computation.

- Binarization should be possible for efficiency without unacceptable degradation of effectiveness (Ahmad and Hawkins, 2015; Zhou et al., 2020).

3.2 Overall Architecture

Figure 1 depicts the overall architecture of our Ultra-High Dimensional (UHD) model. Like most sparse models (e.g., Zamani et al., 2018; Bai et al., 2020), it comprises a query encoder, a document encoder, and a scoring function. While the query and document encoders are run separately, unlike interaction models (e.g., Qu et al., 2021), the weights are shared for the query and document encoders. After the final query and document representations are formed with sparsification and bucketization to be explained below, they are matched with dot product as the scoring function.

The encoder is composed of three modules: 1) the BERT module to convert text into dense token embeddings, 2) the *Winner-Take-All* (WTA) module that sparsifies the BERT module's outputs, and 3) the Max-pool module that performs *non-interfering aggregation* of sparse token embeddings. We define the final output from the WTA module as a *bucket* (Figure 2), implying that the final representation to be used for matching contains

multiple buckets. The BERT and WTA modules are trained jointly with an objective to maximize the similarity between a query bucket and individual relevant document buckets.

Our UHD representations can be seen as bucketed with or segmented into multiple parts that represent different aspects of a query or document. From the procedural point of view, multiple buckets are produced for a query or document by different versions of WTA and concatenated to build the final UHD representation.
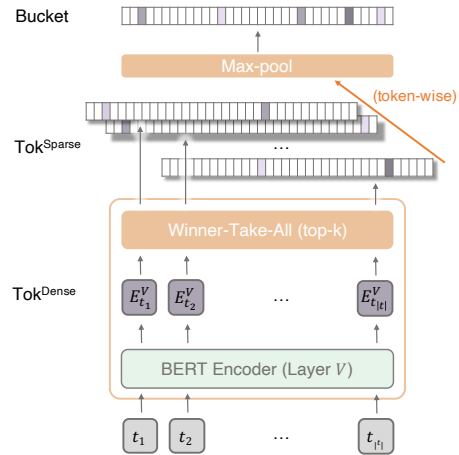
3.3 Query/Document Encoder

Figure 2: The Encoder structure.

Let $q = \{q_1, q_2, ..., q_{|q|}\}$ be a query and $d = \{d_1, d_2, ..., d_{|d|}\}$ be a document. We obtain contextualized dense representations from BERT for the tokens in q and d. Aside from the bucketed representations to be explained in Section 3.5, we here assume a common option of using only the last layer for a query or document representation.

1018

An embedding for q or d is obtained as follows:

$$E_q = BERT(q) \in \mathbb{R}^{|q| \times h}$$
$$E_d = BERT(d) \in \mathbb{R}^{|d| \times h} \quad (1)$$

where h is the hidden size of BERT. Queries and documents are encoded separately with the same encoder.

To sparsify each dense token embedding, we adopt a WTA layer (Makhzani and Frey, 2015), which is an n-dimensional linear layer in which only top-k activations ($k \ll n$) are preserved while the others are set to zero. Let t_i be the ith token of a query or a document and E_{t_i} be its dense embedding. Then a high-dimensional sparse representation S_{t_i} is built as follows:

$$z_{t_i} = E_{t_i} \cdot W + b, \ W \in \mathbb{R}^{h \times n}, \ b \in \mathbb{R}^n \quad (2)$$

$$S_{t_i}[dim] = \begin{cases} z_{t_i}[dim], & \text{if } z_{t_i}[dim] \in \text{top-}k(z_{t_i}) \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

where $dim \in [1, n]$ is dimension of z_{t_i} and S_{t_i}.

In order to drive the learning flow to winning signals, the WTA module considers only the gradients flowing back through the *winning* dimensions. In addition, we impose *weight sparsity* constraint proposed in Ahmad and Scheinkman (2019), which is like applying dropout (Srivastava et al., 2014) individually to output layer's nodes. The benefits of adopting WTA are: 1) We can control sparsity of a resulting embedding precisely and conveniently by adjusting k, an ability considered important for generating sparse representations reliably, and 2) k can be modified at inference time so that the output's sparsity can be altered for an application's need without re-training the model.

Sparse token embeddings generated by the encoder are aggregated with token-wise max-pool followed by L2-normalization to produce a single sparse embedding B_t, a *bucket*:

$$B_t = max\text{-}pool(S_{t_1}, S_{t_2}, ..., S_{t_{|t|}}) \in \mathbb{R}^n \quad (4)$$

Note that our sparse embeddings can be merged with minimal information interference because non-zero dimensions in high-dimensional space are not likely to overlap among the token embeddings, resulting in *non-interfering* max-pooling. This effect is contrasted with dense representations that are prone to lose much information with aggregation. The ability of preserving token-level information, which is critical for IR (Khattab and Zaharia, 2020), makes our method storage efficient.

3.4 Support for Binary Matching and Search

With ultra-high dimensionality, it becomes possible to just count how many *winning signals* of two representations overlap for similarity (binary matching); the probability they accidentally share the same winning signal becomes exponentially smaller with large dimension size (Ahmad and Hawkins, 2015; Ahmad and Scheinkman, 2019). With proper optimization (e.g., (Zhou et al., 2020; Tissier et al., 2019)), it becomes possible to perform highly efficient search while achieving comparable performance. In the experiment section, we empirically show that our model can support the binary matching with negligible impact on the effectiveness.

3.5 Representations with Multiple Buckets

We use multiple buckets to encode information with different levels of abstraction coming from different layers of BERT, as shown in (Jo and Myaeng, 2020). We expect UHD representations extracted from multiple layers contain richer information than from a single layer (e.g., the last layer often used for a downstream task).

We first extract V representations corresponding to the number of the BERT layers for all tokens t.

$$E_t^j = BERT^j(t) \in \mathbb{R}^{|t| \times h} \quad (5)$$

where j is a BERT layer $\in [1, \ V]$. WTA layers are then independently applied to all BERT layers as in Section 3.3 to obtain V buckets.

$$B_t^j = WTA^j(E_t^j) \in \mathbb{R}^n \quad (6)$$

After applying the bucketing mechanism, we obtain B_q^j and B_d^j for q and d so that a relevance score for the query and document is computed by a bucket-wise dot product as follows.

$$Rel(q, d) = \sum_j B_q^j \cdot B_d^j \quad (7)$$

3.6 Training

The entire model is trained to make a query similar to the relevant documents and dissimilar to the irrelevant ones. Given a query q, a set of relevant (positive) documents D^p, and a set of irrelevant (negative) documents D^n, we calculate the loss:

$$\mathcal{L} = \sum_{\substack{d^p \in D^p \\ d^n \in D^n}} \max(0, 1 - Rel(q, d^p) + Rel(q, d^n)) \quad (8)$$

Given a query, we regard the positives of other queries within a mini-batch as negative samples

(i.e., in-batch negatives), following Zhan et al. (2020).

3.7 Ranking

Our model allows for exploiting an inverted index. We regard each dimension in our bucketed sparse representations, which is indexable, as a signal or a latent term. For instance, if n (dimensionality) is 81,920, a document is represented with a combination of a few latent terms out of 81,920. Only the non-zero dimensions of a document enter the inverted index with their weights. The level of efficiency in symbolic IR can be achieved since only a small fraction of the dimension in our UHD representation contains a non-zero value. Even higher efficiency can be gained by using the binarized output for indexing and ranking. For binarization, we convert non-zero values to 1, leaving others as 0.

We first encode all documents in a collection using the trained encoder to construct an inverted index. Each bucket conceptually has its own independent inverted index, resulting in $|B|$ (e.g. the number of BERT layers) inverted indices. Note that this process is needed only once offline. At inference (retrieval) time, we encode a given query to make $|B|$ representations for bucket-wise dot products and sum up the scores for the final relevance score as in Eq. 7. This bucket-wise index construction and scoring can be easily distributed for added efficiency.

4 Experiments

We conduct a series of experiments for validity of the proposed retrieval model and the associated methods against recently proposed sparse methods. We also juxtapose our results with those of the most recent dense models although they are not geared toward full ranking, without resorting to additional computational resources. We defer a qualitative analysis that shows UHD's interpretability to Appendix A.4 due to the lack of space.

While SNRM (Zamani et al., 2018) would be a suitable baseline as a sparse model, we were unable to reproduce the model following the hyperparameter settings presented in the paper. This reproducibility issue is also reported in Medini et al. (2020) and Paria et al. (2020).

Implementation details and hyperparameters are available in Appendix A.1 for reproducibility.

4.1 Settings

Dataset Following previous work, we evaluate on MS MARCO (Bajaj et al., 2018) Passage Retrieval and TREC Complex Answer Retrieval (CAR) (Dietz et al., 2017).

MS MARCO[2] consists of 8.84M passages collected from the Web and 1M queries generated from real-world users of Bing. For training, we use 25% of provided triples, (*query*, *positive passage*, *negative passage*), in all our experiments unless otherwise specified. For evaluation, we use the dev set containing 6,980 queries. We mainly evaluate our model for full ranking, but in order to compare a large number of variants without an excessive computational burden, we also take advantage of the re-ranking setting (marked with †). For evaluation metrics, we use MRR@10 and Recall@K.

TREC CAR[3] is a synthetic dataset, consisting of approximately 29M Wikipedia passages and 3M queries. Following related work (Nogueira et al., 2019; Nogueira and Cho, 2019; Khattab and Zaharia, 2020), we use the first four of five pre-defined folds as a training set, which consists of 5M query-passage pairs. For evaluation, we use the test set (2,254 queries) designated for an official run for TREC-CAR 2017. Our results are in MRR@10 and Mean Average Precision (MAP).

Baselines We include both sparse methods for direct comparisons and dense methods as a reference, which require additional computational resources (e.g., GPU) for ranking.

BM25 (Robertson and Walker, 1994) is a BoW-based sparse method with term weighting based on the query/document frequency signals.

Doc2query (Nogueira et al., 2019) & **DocTTTTTquery** (Nogueira, 2019) is a sparse model that expands documents in the vocabulary space by predicting queries using BERT and T5 (Raffel et al., 2020), respectively.

DeepCT (Dai and Callan, 2020b) is a contextualized term weighting framework, which maps BERT representations to context-aware term weights.

SparTerm (Bai et al., 2020) is a sparse model that generates term-weighted & term-expanded sparse vectors belonging to the vocabulary space.

RepBERT (Zhan et al., 2020) is a dense model that encodes queries and documents separately,

---

[2]Official dataset and evaluation scripts can be found in https://github.com/microsoft/MSMARCO-Passage-Ranking.

[3]We used pre-processed dataset and evaluation scripts available here: https://github.com/nyu-dl/dl4marco-bert

| | MS MARCO | | | | TREC CAR | |
Model	MRR@10	R@50	R@200	R@1000	MRR@10	MAP
Dense Embedding Approaches						
RepBERT	30.4	-	-	94	-	-
ColBERT	36.0	82.9	92.3	96.8	44.3[†]	31.3[†]
BERT Base♠	34.7[†]	-	-	-	-	31.0[†]
BERT Large♠	36.5[†]	-	-	-	-	33.5[†]
Sparse Representation Approaches						
BM25	18.7	59.2	73.8	85.7	-	15.3
Doc2query	21.5	64.4	77.9	89.1	-	18.1
DeepCT	24.3	69	82	91	33.2	24.6
DocTTTTTquery	27.7	75.6	86.9	94.7	-	-
SparTerm	27.94	72.48	84.05	92.45	-	-
UHD-BERT	**30.04**	**77.77**	**88.81**	**96.01**	**37.32**[†]	**25.73**[†]

Table 1: Comparisons on MS MARCO and TREC CAR. The dense approaches suffering from longer latency are shown as a reference. ♠ refers to full-interaction models used for re-ranking. † denotes re-ranking results after BM25 retrieval. Our model under TREC CAR employs re-ranking for comparability with the dense models. The baseline results are from the ColBERT paper, except for RepBERT and SparTerm which are from their own.

with mean-pooling on BERT token embeddings.

ColBERT (Khattab and Zaharia, 2020) is a dense model with token-level late interactions between queries and passages, which are encoded separately.

BERT Base & Large (Nogueira and Cho, 2019) are dense models that fully leverage the interaction between a query and passage. They are often used to re-rank fast BM25 results.

4.2 Overall Effectiveness

Table 1 shows the full-ranking (except for †[4]) performance of the proposed model (UHD-BERT) against the baselines on the two datasets. Among the sparse approaches, UHD-BERT outperforms all the baseline models, from traditional term-based methods to recent neural approaches. Unlike the previous sparse approaches that mainly focus on efficiency at the expense of losing information, our approach achieves richer embeddings using bucketed UHD sparse representations while maintaining the necessary sparsity for efficiency. For this result, we train UHD-BERT with buckets on the layers $\{2, 4, 6, 8, 10, 12\}$, with the dimensionality n being 81,920 and non-zero dimensions k being 80.

The performance of the dense models is also provided in Table 1. Since they are not suitable for full-

ranking without additional efficiency-enhancing measures, the result is only a reference. As expected, they show higher effectiveness than UHD-BERT, due to their heavy interactions between the query and document representations, requiring an inference time overhead. While ColBERT (Khattab and Zaharia, 2020) is most promising with the later interaction idea, it still requires relatively heavy computation, which might be impractical for industrial applications. RepBERT (Zhan et al., 2020) needs to employ external resources (i.e., GPUs) for comparable efficiency. UHD-BERT is on par with RepBERT without such overheads, highlighting the advantage of our model being as efficient as the conventional symbolic IR models and yet approaching to the effectiveness of the dense models requiring heavy interactions.

4.3 Efficiency and Binarized UHD

Since UHD-BERT is designed for full ranking, efficiency is as important as its effectiveness. Due to its sparse nature, efficiency of UHD depends on two factors, n (dimensionality) and k (the number of non-zero dimensions). For retrieval, only the non-zero dimensions of a query are used to search the inverted index. Therefore, the time complexity of the retrieval process is:

$$O(k_q * d_{active}) = O(k_q * k_d/n) \quad (9)$$

where k_q and k_d are k for queries and documents,

---

[4]Re-ranking scores tend to be worse than full-ranking as shown in Khattab and Zaharia (2020)

Model	Complexity
RepBERT	$O(|C| * h)$
ColBERT	$O(|C| * |q| * |d| * h)$
UHD-BERT	$O(|C| * k_q * k_d/n)$

Table 2: Complexity comparison between UHD-BERT and two dense baselines. h denotes BERT dimension size. $n(\gg h)$ and k denote the total and non-zero UHD-BERT dimension sizes, respectively. $|C|$ denotes the collection size. Increasing the dimension size enhances efficiency in UHD-BERT unlike dense models.

Model	Latency(ms)
RepBERT	80[†]
ColBERT	458[†]
BERT Base	-
BERT Large	3,400[†]
BM25	62
Doc2query	85
DeepCT	62 *(est.)*
DocTTTTTquery	87
SparTerm	-
UHD Binarized Inverted Index	63

Table 3: Comparison of latency (ms/query) between our UHD binarized inverted index and the baseline models on the MS MARCO dev set. Baseline results are from the ColBERT paper, except for the RepBERT value from its own paper. † denotes GPU-accelerated document ranking.

respectively. The value of d_{active}, the average number of linked documents per dimension, is approximated by the probability that a dimension becomes non-zero, k_d/n.

Table 2 shows that UHD-BERT has a much lower time complexity than the two dense models, ColBERT and RepBERT, as the actual ratio between the product of k_q and k_d and n is less than h. In fact, the complexity decreases as the dimension size n increases when k is fixed to a constant for limited computing resources. Note, however, that n is also resource-dependent and can account for a trade-off between efficiency and effectiveness together with k as in Section 4.4.

For further improvement in efficiency, we experiment with a binarized UHD-BERT, feasibility of which is justified with the distribution of embedding values. It turns out that the embedding values are clustered into the range between 0.1 and 0.2 as in Figure 3 with the mean and standard deviation being 0.11169 and 0.00182, respectively. This unique pattern assures that binarization can be done without having to coerce the embedding values to either 0 or 1 unnaturally. With binarized UHD embeddings, the retrieval effectiveness remains almost identical (30.03) to the original (30.04) (Table 1).
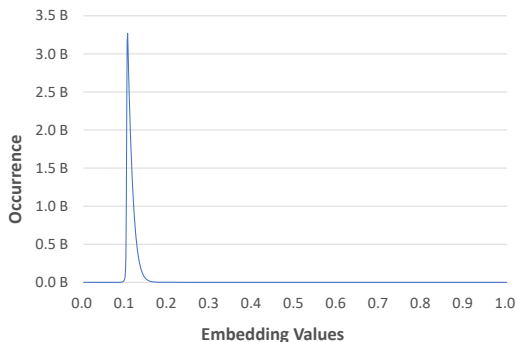
As a result, our experiment on latency is based on an inverted index with a binarized version of UHD-BERT[5]. Table 3 shows comparisons against the baselines for latency, which in our case includes query preprocessing and encoding, and document scoring and sorting. The result clearly shows the efficiency advantage of using the inverted index with UHD-BERT over the baselines; it is almost identical to BM25, known to be extremely efficient.

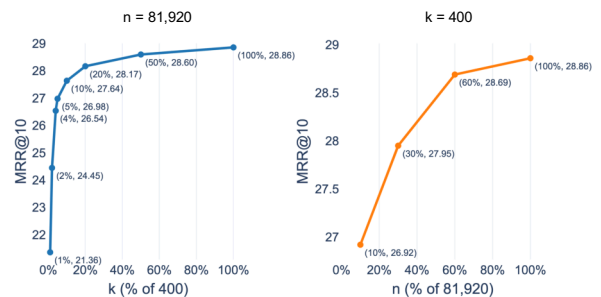## 4.4 Analysis on Dimensionality and Sparsity

Figure 4: Impact of n (dimensionality) and k (non-zero dimensions).

To understand the roles of n (dimensionality) and k (non-zero dimensions), we test our model with two different settings. Figure 4 shows the performance trends: the impact of varying k with {4, 8, 16, 20, 40, 80, 200, 400} on the left using the fixed $n = 81,920$ and the impact of varying n

Figure 3: Distribution of embedding values on UHD.

---
[5]The single bucket option was used but multiple buckets can be easily introduced with server-level parallelism.

with {8,192, 24,576, 49,152, 81,920} on the right using the fixed $k = 400$.

Given the fixed n, it is evident that the higher k, the better the score. This is because the absolute amount of information for representing a query and a document increases. The graph shows, however, that the score rises rapidly to a point and almost stays. It is not desirable to set an exorbitant k (e.g., 8,192) as it would yield high computational costs for a very small gain. We see the score is reasonable even with k being 16 (4%). This suggests that is is important to find an optimal k that satisfies the trade-off between effectiveness and efficiency.

Next, we observe that the score improves as the n increases even with the fixed k. This is because the number of available activation patterns (i.e., expressive power) increases exponentially with larger n. Also, note that larger n with the same k means the increased sparsity (i.e., efficiency). This observation supports the motivation of UHD representations that endow the discrimination power while increasing the efficiency.

Query k	MS MARCO	TREC CAR
50	29.50	34.10
100	**30.04**	35.43
200	29.86	36.31
300	29.64	**37.32**
400	29.29	37.22
Original≈1,200	28.99	37.12

Table 4: MRR@10 with different query k (# of non-zero dimensions of query after max-pool).

Finally, in order to analyze how query's sparsity affects the performance, we constrain each query's final k (query k) after the max-pool operation and measure MRR@10, expecting a removal of trivial information that can cause query-drift. Table 4 shows how MRR@10 changes as we change the query k. The highest performance is achieved with k being much lower than the original ($\approx$1,200). This means removing trivial information indeed helps improve matching quality. Another huge benefit is that it reduces the amount of required computation for matching.

4.5 Analysis on Multiple Buckets

In order to verify the effectiveness of our bucketing strategy, we compare the performance of single and multi-bucket strategies in Table 5. For a fair

Strategy	MRR@10
Single-Bucket ($b = 1$, $n = 49,152$, $k = 480$)	26.18
Multi-Bucket ($b = 6$, $n = 8,192$, $k = 80$)	27.08
+ bucket weight tuning	**27.41**

Table 5: Comparison of bucketing strategies on the MS MARCO dev set re-ranking task. The total dimensionality (n) and non-zero dimension size (k) are the same for both strategies. b stands for the number of buckets.

comparison, we set the total dimensionality (n) and the number of non-zero dimensions (k) to be the same for both of them. For the mutliple bucket setting, we create each bucket with the 2nd, 4th, 6th, 8th, 10th, and 12th layers of BERT, whereas the 12th layer of BERT is used for the single-bucket setting. For training, we only use 3% of the MS MARCO Train Triples Small set.

Our multi-bucket strategy is shown to give a good improvement, and the bucket weighting scheme improves it further. This result validates the idea explained in Section 3.5 that it is crucial for IR tasks to exploit low- and mid-level lexical and syntactic information, as well as the last layer containing well-refined semantics. We provide additional analyses on other settings in Appendix A.3.

5 Conclusion

We present UHD-BERT, a novel retrieval method empowered by extremely high dimensionality and sparsity that is easily controllable. To fully utilize the capacity of our representation scheme, we propose a bucketing mechanism that incorporates different linguistic aspects extracted from BERT and achieves the goal of building a neural retrieval model that ameliorates the synonymy and polysemy problems of symbolic text retrieval methods. With binarization of the learned UHD representations and resulting inverted indices, we attain the desired efficiency that is comparable with BM25 and hence suitable for industrial applications. The benefits of the UHD representations and the resulting retrieval model were demonstrated with two different IR datasets, in comparison with the recent approaches to sparse representation methods. We also show that our method has a significant advantage in efficiency over the state-of-art dense models, with a slightly lower but competitive effectiveness.

We plan to investigate how the query-document interactions and symbolic IR techniques can be incorporated for further improvements.

Acknowledgements

References

Subutai Ahmad and Jeff Hawkins. 2015. Properties of sparse distributed representations and their application to hierarchical temporal memory. *arXiv preprint arXiv:1503.07469*.

Subutai Ahmad and Luiz Scheinkman. 2019. How can we be so dense? the benefits of using highly sparse representations. *CoRR*, abs/1903.11257.

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. 2018. Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics*, 6:483–495.

Hiteshwar Kumar Azad and Akshay Deepak. 2019. Query expansion techniques for information retrieval: a survey. *Information Processing & Management*, 56(5):1698–1735.

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

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.

Yonatan Belinkov and James Glass. 2019. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.

Zhuyun Dai and Jamie Callan. 2020a. Context-aware document term weighting for ad-hoc search. In

WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020, pages 1897–1907. ACM / IW3C2.

Zhuyun Dai and Jamie Callan. 2020b. Context-aware term weighting for first stage passage retrieval. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 1533–1536. ACM.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Laura Dietz, Manisha Verma, Filip Radlinski, and Nick Craswell. 2017. Trec complex answer retrieval overview. TREC.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A. Smith. 2015. Sparse overcomplete word vector representations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1491–1500, Beijing, China. Association for Computational Linguistics.

Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128.

Jae-young Jo and Sung-Hyon Myaeng. 2020. Roles and utilization of attention heads in transformer-based neural language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3404–3417, Online. Association for Computational Linguistics.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

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*, pages 39–48. ACM.

Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2021. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345.

Alireza Makhzani and Brendan J. Frey. 2015. Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2791–2799.

Tharun Medini, Bei di Chen, and Anshumali Shrivastava. 2020. Solar: Sparse orthogonal learned and random embeddings. *ArXiv*, abs/2008.13225.

Rodrigo Nogueira. 2019. From doc2query to docttttt-query.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

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

Biswajit Paria, Chih-Kuan Yeh, Ian En-Hsu Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. 2020. Minimizing flops to learn efficient sparse representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

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*, pages 5835–5847, Online. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

Nils Reimers and Iryna Gurevych. 2020. The curse of dense low-dimensional information retrieval for large index sizes. *arXiv preprint arXiv:2012.14210*.

Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94*, pages 232–241. Springer.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. 2016. Sparse word embeddings using l1 regularized online learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2915–2921.

Julien Tissier, Christophe Gravier, and Amaury Habrard. 2019. Near-lossless binarization of word embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7104–7111.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

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*, pages 497–506. ACM.

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2020. Repbert: Contextualized text embeddings for first-stage retrieval. *arXiv preprint arXiv:2006.15498*.

Lei Zhou, Xiao Bai, Xianglong Liu, Jun Zhou, and Edwin R Hancock. 2020. Learning binary code for fast nearest subspace search. *Pattern Recognition*, 98:107040.

A Appendix

A.1 Experimental Setting Details

Implementation For most of our experiments, the language model used in our architecture is the official pre-trained BERT (*bert-base-uncased*) (Devlin et al., 2019) by Hugging Face (Wolf et al., 2020). Only for the TREC CAR experiment, we used a different pre-trained model as in Khattab and Zaharia (2020), which is the pre-trained *Large* model released by Nogueira and Cho (2019) who ensures that the test set of TREC CAR is not exposed for pre-training. Since TREC CAR is based on Wikipedia, Nogueira and Cho (2019) pre-trained BERT on the subset of Wikipedia so that the model is not exposed to the test set of TREC CAR during the pre-training step. They released their pre-trained *Large* model and we fine-tune it for TREC CAR experiment. Fine-tuning this model is remarkably slower than BERT *Base*, we leverage the single layer (23) of BERT *Large* to report the result of TREC CAR. For the buckets on top of the LM, we modified Winner-Take-All (WTA) layer (Makhzani and Frey, 2015) by adding a weight sparsity proposed by Ahmad and Scheinkman (2019). Our UHD model has about 172M parameters per bucket with $n = 81,920$ for MS MARCO and 398M for TREC CAR. For the binarized inverted index, we use a bit packing compression for memory efficiency. Since compression is not necessary in an environment with sufficient memory, we excluded the time consuming due to compression in the latency measurement.

Training We use 25% of MS MARCO train triples, (*query*, *positive passage*, *negative passage*), containing a total of 39,782,779 triples, in all our experiments unless otherwise specified. Note that we use only queries and positive passages with in-batch negatives for training, ignoring the negative passages. We sample 7,000 queries that are not in the above 25% to construct a re-ranking train subset with the corresponding top 1000 passages, provided by MS MARCO. This train subset is used for the bucket weight tuning in Section 4.5. For TREC CAR, we use the first four of five pre-defined folds as a training set which consists of query-passage pairs (approximately 5M pairs), following related works (Nogueira et al., 2019; Nogueira and Cho, 2019; Khattab and Zaharia, 2020). We utilized dataset pre-processed and provided by Nogueira and Cho (2019). Training a single-bucket model

from the last layer of BERT with $n = 81,920$ (dimensionality) and $k = 80$ (non-zero dimensions) takes 15.5 hours on two GeForce RTX 3090 devices for MS MARCO and 40.86 hours on a same single device for TREC CAR.

Evaluation MS MARCO provides two settings: re-ranking and full ranking. In the re-ranking setting, given a query and 1000 passages (top-1000) retrieved by BM25, the model should re-rank the passages. In full ranking, on the other hand, the model should retrieve relevant passages from the entire collection for a given query. While our main goal is to validate our model in the full ranking setting, we utilize the re-ranking setting when a large number of model variants are compared, as a way to save computational cost. In both settings we use the MS MARCO dev set, which contains 6,980 queries. MRR@10 (mean reciprocal rank) is used as the evaluation metric as in the official evaluation. We use recall at various levels as additional metric. For TREC CAR, we use the same test set (2,254 queries) used to submit to TREC-CAR 2017. Following Nogueira et al. (2019); Khattab and Zaharia (2020), we report re-ranking results after BM25 retrieval for fair comparison, using MRR@10 as well as the official metric, MAP (Mean Average Precision). Single RTX 3090 was used for encoding queries and documents, and re-ranking. For full ranking using a binarized inverted index, we use a Threadripper 3960X (CPU) with 24 cores.

Hyperparameters We use the Adam optimizer with the learning rate of 5e-6, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and epsilon = 1e-8, and L2 weight decay of 0.01. The learning rate is linearly warmed up over the first 2,000 steps, and then linearly decayed. The batch size is 32, and the epochs are 0.25 for MS MARCO (25% of MS MARCO Train Triples Small set) and 1.0 for TREC CAR (all query-passage pairs in the four folds of TREC CAR). For the BERT LM, we use GELU (Hendrycks and Gimpel, 2017) as an activation and set the maximum length to 32 for queries and 180 for passages (truncated if exceeded). For the WTA layer, we select dimensionality n from {8,192, 24,576, 49,152, 81,920} and the number of non-zero dimensions k from {4, 8, 16, 20, 40, 80, 200, 400}. For the analysis of n and k, we once trained with the k of 400 and changed it to above settings at test time. There was no significant difference between the setting where k matches and the setting that does
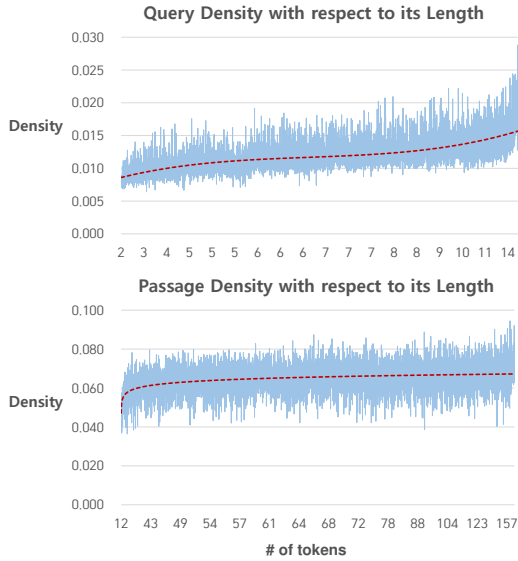
Figure 5: Density of the generated sparse representation of queries and passages. Queries and passages are sorted according to their token lengths (x-axis).

not match at training and test times. After having 3 times of trials for each setting, our final setting is $n = 81,920, k = 80$, matching the balance between the performance (MRR@10) and efficiency. We set weight sparsity to 0.7 from $\{0.3, 0.5, 0.7\}$, meaning that 70% of the weights connecting each WTA node to input is set to zero.

A.2 Analysis on UHD Embeddings

To better understand the characteristics of our proposed model, we conducted an in-depth analysis of the generated embeddings. In this experiment, we use a model with a single bucket on the 12th layer of BERT LM, where dimensionality n and the number of non-zero dimensions k are 81,920 and 80, respectively.

In order to check whether the generated embedding properly contains the information of queries and passages, in Figure 5, we measured the density (ratio of non-zero dimension) of the queries (left-hand side) and passages (right-hand side) with respect to their length. Intuitively, we can assume that the more words (or tokens) the text contains, the more information it contains. Because each dimension of sparse embedding is activated only when the corresponding feature exists, the density of the generated embeddings must correlate with the amount of information (the number of tokens) in the query/passage, as shown in Figure 5. However, interestingly, the density continues to increase according to the number of tokens in the query,

while it converges in the passage. Our interpretation is that since the query reflects the intention of the user, it is highly likely that each query term represents a unique feature that narrows down the search space, which does not overlap with other query terms. For the example *'how much do psychologists earn uk'*, a user has put *'uk'* to reduce the search space. As such, increasing the length of the query has a high probability of adding new features, which increases the density. Unlike queries, passages usually deal with limited topics, which makes density converges even if the length increases.
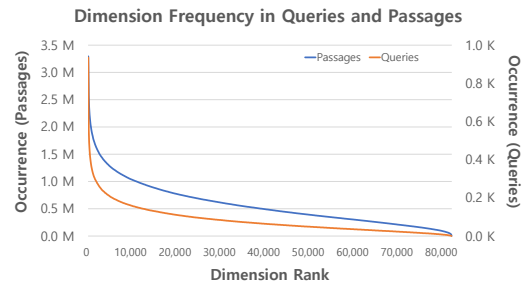

Figure 6: Active (non-zero) dimension frequency in queries and passages.

Besides the above query/passage-level embedding analysis, dimension-level analysis is also crucial to understand the characteristics of our proposed model. In Figure 6, we measured how many times each dimension was activated (non-zero) in the query and passage, similar to traditional word frequency analysis. The analysis suggests that the embeddings generated by our model work in a similar fashion to words in natural languages (Zipf's law). It is also worth noting that similar to stop words in natural language words, there are dimensions that appear in many passages (8.84M in total) and queries (7K in total). Dealing with these "stop-dimensions" can also be expected to have a big impact on performance, but we leave it as future work.

A.3 Additional Analysis on Multiple Buckets

In addition to the method and analysis on multiple buckets (Section 3.5, 4.5), here we address further various settings and their analysis.

For the multiple bucket setting, we evaluate two types of implementation (Joint/Separate). The joint setting means that we train all WTA layers in an end-to-end manner, while the separate setting denotes that we separately train the model for each WTA layer and combine the results later. We notice

1027

Strategy	MRR@10
Single-Bucket $(b = 1, n = 49,152, k = 480)$	26.18
Multi-Bucket $(b = 6, n = 8,192, k = 80)$	
Multiple bucket (Joint)	25.58
Multiple bucket (Separate)	27.08
+ bucket weight tuning	**27.41**
Ideal query-layer predictor	**39.62**

Table 6: Comparison of various multi-bucket strategies on the MS MARCO Passage Retrieval dev set re-ranking task. The total dimensionality (n) and non-zero dimension size (k) are the same across the strategies. b stands for the number of buckets.

that the joint setting of the multi-bucket unexpectedly underperforms the single-bucket. We speculate that the WTA layers in the joint setting easily interfere with each other during training, implying the need for a special mechanism to handle the interference for future work. On the other hand, the separate setting, which is free from the interference, outperforms the single bucket setting, despite the disadvantages of the multi-bucket strategy (small n for each bucket). This is also evident in the result of the bucket weight tuning. By controlling the weight of each bucket, we can further increase the performance of the multiple bucket setting.

For the bucket weight tuning, we conducted a grid search on the top-1000 re-ranking dataset we built from the MS MARCO dataset (See Section A.1), which yields the searched weights (0.66 : 0.0 : 0.33 : 1.00 : 0.33 : 1.00).

Going further from this analysis, one can imagine that each query may have a more suitable set of layer weights. For example, certain queries may require syntactic contents while others may require topical (semantic) contents. From this perspective, we measured the performance when a layer with the highest MRR was ideally selected for each query (Ideal query-layer predictor in Table 6). The result clearly suggests that selecting different layers or bucket weight distributions for each query has the potential to improve performance. This implies the necessity of a bucket weight predictor which takes a query as an input, but we leave it as future work.

A.4 Qualitative Analysis on Interpretability

One of the advantages of the sparse representation is its interpretability, which is possible in our case due to the disentangled feature of the UHD embedding. To prove this, we conduct an analysis about the meaning of each dimension of the generated UHD embedding. In this analysis, we use the UHD-BERT model with reduced $k(= 40)$, to reduce the active dimension for making the analysis easier.

Table 7 shows interpretation results on a few sampled dimensions of the UHD embedding. We use a simple method that validates the interpretation, which is to inspect terms of queries that have non-zero values (activated) on a specific dimension. For example of the dimension index "61,183" in Table 7, the terms that activated queries frequently have were *child*, *kids* and *disney*, implying that the dimension is specialized for the "child" information. For clarity, query terms that have appeared at least five times are listed, along with dimensions that are relatively easy to interpret. We observe that there are dimensions representing abstract latent terms such as "Unit" or "Period". To summarize the analysis, our UHD embedding is not only interpretable (advantage of sparse representations), but also has the power of the latent representation (advantage of dense representations).

Dim	Appeared Query Terms	Interpretation
68,604	*costs*, *paid*, *fee*, *prices*, *fees*	Price
29,938	*tall*, paid, *square*, *feet*, *miles*, *words*	Unit
8,797	*soon*, *longest*, heal, *wait*, *been*, *length*, *weeks*,	Period
66,589	*growth*, *movement*, *development*, degree, theory	Advancement
3,740	*treatment*, *heal*, *surgery*, *doctors*, *doctor*	Treatment
24,673	*medication*, normal, *constipation*, *doctor*, *medicine*	Medical
37,089	*problems*, *diseases*, *signs*, *syndrome*, *fever*	Disease
476	*die*, *killed*	Death
47,873	*technology*, *science*, *chemical*, *scientific*, union	Science
374	science, technology, fastest, *java*, *language*, *software*	Programming
68,111	lower, low, clear, *deductible*, *irs*, depression	Tax
21,822	*party*, beach, *amendment*, tissue, *constitution*	Politic & Law
22,627	*founded*, *invented*, *begin*, *established*, *started*, released	Founding
3,266	*numbers*, *customer*, *telephone*, *zip*, *contact*, union, npi	E-commerce
46,843	*washington*, *il*, *ma*, *mn*, *indiana*, *south*	Location
61,183	*child*, *kids*, *children*, *disney*, paint, *parents*	Child
48,698	*education*, *student*, training, *degree*, workout	Education
26,433	*synonym*, *description*, *synonyms*, *defined*, english	Definition
1,377	*sang*, *played*, *sings*, requirements, *cats*, *plays*	Play (theatre)

Table 7: Interpretation of the each dimension in the UHD embedding by analyzing terms of queries that activates a specific dimension. The query terms that support the interpretation are indicated in *italic*.