SDR: Efficient Neural Re-ranking using Succinct Document Representation

Nachshon Cohen*
Amazon
nachshon@amazon.com

Amit Portnoy*
Ben-Gurion University†
amitport@post.bgu.ac.il

Besnik Fetahu
Amazon
besnikf@amazon.com

Amir Ingber
Pinecone Systems†
ingber@pinecone.io

Abstract

BERT based ranking models have achieved superior performance on various information retrieval tasks. However, the large number of parameters and complex self-attention operations come at a significant latency overhead. To remedy this, recent works propose late-interaction architectures, which allow precomputation of intermediate document representations, thus reducing latency. Nonetheless, having solved the immediate latency issue, these methods now introduce storage costs and network fetching latency, which limit their adoption in real-life production systems.

In this work, we propose the Succinct Document Representation (SDR) scheme that computes *highly compressed* intermediate document representations, mitigating the storage/network issue. Our approach first reduces the dimension of token representations by encoding them using a novel autoencoder architecture that uses the document's textual content in both the encoding and decoding phases. After this token encoding step, we further reduce the size of the document representations using modern quantization techniques.

Evaluation on MSMARCO's passage re-ranking task show that compared to existing approaches using compressed document representations, our method is highly efficient, achieving 4x–11.6x higher compression rates for the same ranking quality. Similarly, on the TREC CAR dataset, we achieve 7.7x higher compression rate for the same ranking quality.

1 Introduction

Information retrieval (IR) systems traditionally comprise of two stages: retrieval and ranking. Given a user query, the role of the retrieval stage is to quickly retrieve a set of candidate documents from a (very large) search index. Retrieval is typically fast but not accurate enough; in order to improve the quality of the end result for the user, the candidate documents are re-ranked using a more accurate but computationally expensive algorithm.

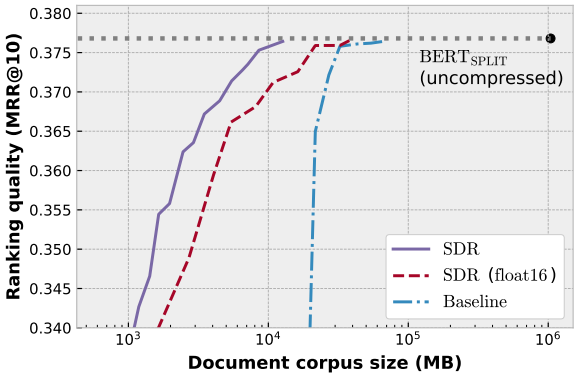

Figure 1: MRR@10 performance vs. document corpus size tradeoff, measured on the MSMARCO-DEV dataset. $\text{BERT}_{\text{SPLIT}}$ is a distilled late-interaction model with reduced vector width and no compression (§ 4.2). For MRR@10 above 0.35, SDR is 4x–11.6x more efficient compared to the baseline.

Neural approaches have achieved the state of the art ranking performance in IR applications (Yates et al., 2021). Transformer networks such as BERT (Devlin et al., 2019) consistently show better ranking effectiveness at the cost of a higher computational cost and latency (Nogueira and Cho, 2019).

To rank k documents, the ranker is called k times with an input of the form (query, document), where the query is the same, but the document is different. Several works (MacAvaney et al., 2020; Gao et al., 2020b; Chen et al., 2020; Cao et al., 2020; Nie et al., 2020; Gao et al., 2020b; Khattab and Zaharia, 2020) have proposed to modify BERT-based rankers in a way that allows part of the model to compute query and document representations separately, and then produce the final score using a low-complexity interaction block; we denote these models as late-

*Both authors contributed equally to the paper.
†Work carried out while working at Amazon.

interaction rankers. Such approaches pre-compute document representations to improve latency significantly. Next, at runtime the model computes the query representation (once), retrieves the pre-computed document representations, and is only required to run a low-complexity interaction block k times to produce the final ranking score.

Precomputing document representations has shown to significantly reduce latency and at the same time retain comparable scores to BERT models (Gao et al., 2020b). However, this does not account for additional storage and/or network fetching latency costs. The representations typically consist of the contextual token embeddings in a transformer model, which consume orders of magnitude more storage than storing the entire corpus search index (cf. § 5.1).

In this work, we propose Succinct Document Representation (SDR), a general scheme for compressing document representations. It enables late-interaction rankers to be efficient in both latency and storage, while maintaining high ranking quality. SDR is suitable for any ranking scheme that uses contextual embeddings, and achieves extreme compression ratios (2-3 orders of magnitude) with little to no impact on retrieval accuracy. SDR consists of two major components: (1) embedding dimension reduction using an *autoencoder* with *side information* and (2) *distribution-optimized quantization* of the reduced-dimension vectors.

In SDR, the autoencoder consists of two subnetworks: an encoder that reduces the vector's dimensions and a decoder that reconstructs the compressed vector. The encoder's output dimension represents the tradeoff between reconstruction fidelity and storage requirements. To improve the compression-reliability tradeoff, we leverage *static* token embeddings, which are available since the ranker has access to the document text (as it needs to render it to the user), and are computationally cheap to obtain. We feed these embeddings to both the encoder and decoder as *side information*, allowing the autoencoder to focus more on storing "*just the context*" of a token, and less on its original meaning that is available in the *static* embeddings. Ablation tests verify that adding the static vectors significantly improves the compression rates for the same ranking accuracy.

Since data storage is measured in bits rather than floating-point numbers, SDR uses quantization techniques to reduce storage size further. Given that it is hard to evaluate the amount of information in each of the encoder's output dimensions, we perform a randomized Hadamard transform on the vectors, resulting in (1) evenly spread information across all coordinates and (2) transformed vectors that follow a Gaussian-like distribution. We utilize known quantization techniques to represent these vectors using a small number of bits, controlling for the amount of quantization distortion.

Existing late-interaction schemes either ignore the storage overhead, or consider basic compression techniques, such as a simple (1 layer) autoencoder and float16 quantization. However, this is insufficient to reach reasonable storage size (MacAvaney et al., 2020); furthermore, this results in an increased fetching latency. For the MSMARCO dataset, we used a distilled model with a reduced vector width (Hofstätter et al., 2020a) as the initial pre-trained weights for the late-interaction model. On top of this, we used a non-linear autoencoder consisting of 2 dense layers followed by float16 quantization, a natural extension of MacAvaney et al. (2020). This baseline achieves compression rates of 30x with no noticeable reduction in retrieval accuracy (measured with the official MRR@10 metric). In comparison with this strong baseline, our SDR scheme achieves an additional compression rate of between 4x to 11.6x with the same ranking quality, reducing document representation size to the same order of magnitude as the retrieved text itself. In Figure 1 we include a high-level presentation of the baseline, a variant of our method with float16 quantization, and our full method. For the TREC CAR dataset, for which we do not have a reduced-width baseline, we used a BERT model as the pre-trained weights for the late-interaction model. The baseline with 2 dense layers and float16 quantization achieves a 30x compression rates with a slight reduction in accuracy. The SDR scheme reaches the same quality while improving compression rate by another 7.7x.

To summarize, here are the contribution of this work[1]:

- We propose the Succinct Document Representation (SDR) scheme for compressing the document representations required for fast Transformer-based rankers. The scheme is based on a specialized autoencoder architecture and subsequent quantization.

[1]Code is available at https://github.com/amzn/amazon-succinct-doc-representation

- For the MSMARCO passage retrieval task, SDR shows compression ratios of 121x with no noticeable decrease in ranking performance. Compared to existing approaches for producing compressed representations, our method attains better compression rates (between 4x and 11.6x) for the same ranking quality. Similar results are demonstrated on the TREC CAR dataset.

- We provide a thorough analysis of the SDR system, showing that the contribution of each of the components to the compression-ranking effectiveness is significant.

2 Related Work

Late-interaction models. The idea of running several transformer layers for the document and the query independently, and then combining them in the last transformer layers, was developed concurrently by multiple teams: PreTTR (MacAvaney et al., 2020), EARL (Gao et al., 2020a), DC-BERT (Nie et al., 2020), DiPair (Chen et al., 2020), and the Deformer (Cao et al., 2020). These works show that only a few layers where the query and document interact are sufficient to achieve results close to the performance of a full BERT ranker at a fraction of the runtime cost. For each document, the contextual token vectors are stored in a cache and retrieved during the document ranking phase. This impacts both storage cost as well as latency cost of fetching these vectors during the ranking phase. MORES (Gao et al., 2020b), extends late-interaction models, where in the last interaction layers only the query attends to the document (and not vice-versa). As document are typically much longer, this results in additional performance improvements with similar storage requirements. ColBERT (Khattab and Zaharia, 2020) is another variant that runs all transformer layers independently for the query and the document, and the interaction between the final vectors is done through a sum-of-max operator. A similar work, the Transformer-Kernel (TK) (Hofstätter et al., 2020b), has an interaction block based on a low-complexity kernel operation. Both ColBERT and TK result in models with lower runtime latency at the expense of a drop in ranking quality. However, the storage requirements for both approaches are still significant.

Some of the works above acknowledge the issue of storing the precomputed document representations and proposed partial solutions. In ColBERT (Khattab and Zaharia, 2020), the authors proposed to reduce the dimension of the final token embedding using a linear layer. However, even moderate compression ratios caused a large drop in ranking quality. In the PreTTR model (MacAvaney et al., 2020), it was proposed to address the storage cost by using a standard auto-encoder architecture and the float16 format instead of float32. Again, the ranking quality drops even with moderate compression ratios (they measured up to 12x).

Several other works (Guu et al., 2020; Karpukhin et al., 2020; Xiong et al., 2021; Qu et al., 2020; Lu et al., 2020) proposed representing the queries and documents as vectors (as opposed to a vector per token), and using dot product as the interaction block. While this ranker architecture approach is simple (and can also be used for the retrieval step via an approximate nearest neighbor search such as FAISS (Johnson et al., 2017), ScaNN (Guo et al., 2020) or the Pinecone managed service[2]), the overall ranking quality is generally lower compared to methods that employ a query-document cross-attention interaction. For that reason these methods are used mainly for first-stage retrieval, followed by a reranking step.

Compressed embeddings. Our work reduces storage requirements by reducing the number of bits per floating-point value. Quantization gained attention and success in reducing the size of neural network parameters (Gupta et al., 2015; Essam et al., 2017; Wang et al., 2018; Wu et al., 2018) and distributed learning communication costs (Suresh et al., 2017; Alistarh et al., 2017; Konečnỳ and Richtárik, 2018; Vargaftik et al., 2021, 2022). Specifically, compressing word embeddings has been studied as an independent goal. May et al. (2019) studied the effect of quantized word embeddings on downstream applications and proposed a metric for quantifying this effect with simple linear models that operate on the word embeddings directly. As our work is concerned with compressing *contextual* embeddings, these methods do not apply since the set of possible embeddings values is not bounded by the vocabulary size. Nevertheless, as in (May et al., 2019), we also observe that simple quantization schemes are quite effective. Our work uses recent advances in this area to further reduce storage requirements for document representation, which, to the best of our knowledge, were not previously attempted in this context.

[2] www.pinecone.io

3 Succinct Document Representation (SDR)

Our work is based on the late-interaction architecture (MacAvaney et al., 2020; Gao et al., 2020b; Chen et al., 2020; Cao et al., 2020; Nie et al., 2020), which separates BERT into L independent layers for the documents and the queries, and $T - L$ interleaving layers, where T is the total number of layers in the original model, e.g., 12 for BERT-Base. Naively storing all documents embeddings consumes a huge amount of storage with a total of $m \cdot h \cdot 4$ bytes per document, where m is the average number of tokens per document and h is the model hidden size (384 for the distilled version and 768 for the BERT version). For MSMARCO, with 8.8M documents and $m = 76.9$, it leads to a high storage cost of over a terabyte, which is not affordable except in large production systems.

Our compression scheme for the document representations consists of two sequential steps, (i) dimensionality reduction and (ii) block-wise quantization, described in § 3.1 and § 3.2 respectively.

3.1 Dimensionality Reduction using AutoEncoders with Side Information (AESI)

To compress document representations, we reduce the dimensionality of token representations (i.e., the output of BERT's L-th layer) using an autoencoder. Standard autoencoder architectures typically consist of a neural network split into an encoder and a decoder: the encoder projects the input vector into a lower-dimension vector, which is then reconstructed back using the decoder.

Our architecture, AESI, extends the standard autoencoder by using the document's text as *side information* to both the encoder and decoder. Such an approach is possible since, no matter how the document scores are computed, re-ranking systems have access to the document's text in order to render it back to the user. In the rest of this section, we add the precise details of the AESI architecture.

Side Information. In line with our observation that the ranker has access to the document's raw text, we propose utilizing the *token embedding* information, which is computed by the embedding layer used in BERT's architecture. The token embeddings encode rich semantic information about the token itself; however, they do not fully capture the context in which they occur; hence, we refer to them as *static embeddings*. For example, through token embeddings, we cannot disambiguate between the different meanings of the token `bank`, which can refer to either a geographical location (e.g., "river bank") or a financial institution, depending on the context.

Static embeddings are key for upper BERT layers, which learn the contextual representation of tokens via the self-attention mechanism. We use the static embeddings as side information to both the encoder and decoder parts of the autoencoder. This allows the model to focus on encoding the *distilled context*, and less on the token information since it is already provided to the decoder directly.

AESI Approach. For a token whose representation we wish to compress, our approach proceeds as follows. We take the L-th layer's output contextual representation of the token together with its static embedding and feed both inputs to the autoencoder. The information to be compressed (and reconstructed) is the contextual embedding, and the side-information, which aids in the compression task, is the static embedding. The decoder takes the encoder output, along with the static embedding, and attempts to reconstruct the contextual embedding. Figure 2 shows the AESI architecture.

AESI approach has two parameters that are determined empirically. First, the L-th transformer layer of the contextual representation provided as input, which has a direct impact on latency[3]. Second, the size of the encoder's output directly impacts the compression rate and thus storage costs.

Encoding starts by concatenating the input vector (i.e., the output of layer L, the vector we compress) and the static token embedding (i.e., the output of BERT's embedding layer), and then passes the concatenated vector through an encoder network, which outputs a c-dimensional *encoded vector*. Decoding starts by concatenating the encoded vector with the static token embedding, then passes the concatenated vector through a decoder layer, which reconstructs the input vector. Specifically, we use a two-layer dense network for both the encoder and the decoder, which can be written using the following formula:

$$e = E(v, u) := W_2^e \cdot \big(\text{gelu}\big(W_1^e(v; u)\big)\big) \quad (1)$$

$$v' = D(e, u) := W_2^d \cdot \big(\text{gelu}\big(W_1^d(e; u)\big)\big) \quad (2)$$

where $v \in \mathbb{R}^h$ is the contextualized token em-

[3] A ranker has to compute layers $L + 1$ onward online.

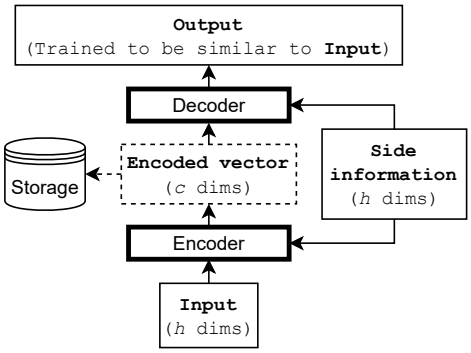

Figure 2: AutoEncoder with Side Information (AESI) architecture. For our usage, the input is the contextual token embedding (the L-th layer's output), and the side information is the static token embedding (the output of BERT's initial embedding layer). The resulting c–dimensional encoded vector can be thought of as the distilled context of the input token.

bedding (the output of the L-th layer), $u \in \mathbb{R}^h$ is the static token embedding (the output of the embedding layer, which is the input to BERT's layer 0 and includes token position embeddings and type embeddings), and $u; v$ means concatenation of these vectors. $W_1^e \in \mathbb{R}^{i \times 2h}$, $W_2^e \in \mathbb{R}^{c \times i}$, $W_1^d \in \mathbb{R}^{i \times (c+h)}$, $W_2^d \in \mathbb{R}^{h \times i}$ are trainable parameters. h is the dimension of token embeddings (e.g., 384), i is the intermediate autoencoder size, and c is the dimension of the projected (encoded) vector. $\mathrm{gelu}(\cdot)$ is an non-linear activation function (Hendrycks and Gimpel, 2016). Additional autoencoder variations are explored in § 5.3.

3.2 Quantization

Storing the compressed contextual representations in a naive way consumes 32 bits (float32) per coordinate per token, which is still costly. To further reduce storage overhead, we propose to apply a quantization technique, which uses a predetermined B bits per coordinate. However, different coordinates and different tokens have different importance and possibly also different scales, so using the same number of bits and same quantization threshold for all of them increases the quantization error.

To remedy this issue, we follow an approach similar to EDEN quantization (Vargaftik et al., 2022), which uses a randomized Hadamard transform prior to quantization. Loosely speaking, this shuffles the information across all coordinates. Furthermore, each of the coordinates is guaranteed to follow Gaussian-like distribution, for which quantization boundaries can be computed optimally. For the sake of brevity, the full description of the quantization algorithm is deferred to Appendix A.

Efficiently applying the Hadamard transform requires the size of the input to be a power of two. In addition, the input dimension should be large enough (specifically, larger than the output of AESI) so that information can be shuffled effectively. Therefore, we concatenate the AESI vectors of all tokens from a single document, then segment it to a larger block size (we use 128), padding the last block with zeros when necessary. The padding slightly increases space requirements and is considered when evaluating the compression efficiency.

4 Experimental Settings

In this section we describe the datasets used to evaluate the competing approaches for ranking documents given a query. Next, we describe the baseline and the different configurations of SDR with emphasis on how we measure the compression ratio.

4.1 Tasks and Datasets

To evaluate the effectiveness of our proposed approach (SDR) and the competing baseline, we consider two information retrieval datasets, each with different characteristics.

MSMARCO passage re-ranking In this task (Nguyen et al., 2016), we are given a query and a list of 1,000 passages (retrieved via BM25), and the task is to rerank the passages according to their relevance to the query. The corpus consists of 8.8M passages, downloaded from the web. We consider two query sets:

(1) *MSMARCO-DEV*, the development set for the MSMARCO passage reranking task, which consists of 6,980 queries. On average, each query has a single relevant passage, and other passages are not annotated. The models are measured using the mean reciprocal rank metric (MRR@10).

(2) *TREC 2019 DL Track*. Here we consider the test queries from TREC 2019 DL Track passage reranking dataset. Unlike MSMARCO-DEV, there are multiple passages annotated for each query with graded relevance labels (instead of binary labels), allowing us to use the more informative nDCG@10 metric. Due to the excessive annotation overhead, this dataset consists of just 200 queries, so results are noisier compared to MSMARCO-DEV.

TREC Complex Answer Retrieval (CAR) is a dataset (Dietz et al., 2017) curated from Wikipedia. It maps from article and section titles to relevant paragraphs. Following Nogueira and Cho (2019),

we use the automatic by-article annotations variant, which considers all paragraphs within the same article as relevant. The dataset consists of 30M passages, making storage requirements a more significant challenge compared to the MSMARCO task. The test query set consists of 2,254 queries with an average of 2.74 positive passages per query. We use the MAP@1K official metric.

For both datasets, in addition to the quality metrics, we also measure the *Compression Ratio* (CR) as the amount of storage required to store the token embeddings when compared to the baseline model. E.g., CR $= 10$ implies storage size that is one tenth of the baseline vectors.

4.2 Baseline – *BERT*$_{\text{SPLIT}}$

Our algorithm is based on the late-interaction architecture (MacAvaney et al., 2020; Gao et al., 2020a; Nie et al., 2020; Chen et al., 2020; Cao et al., 2020). We created a model based on this architecture, which we name BERT$_{\text{SPLIT}}$, consisting of 10 layers that are computed independently for the query and the document with an additional two late-interaction layers that are executed jointly. For MSMARCO, we initialized the model from reduced width pre-trained weights[4] and fine-tuned it using knowledge distillation from an ensemble of BERT-Large, BERT-Base, and ALBERT-Large (Hofstätter et al., 2020b) on the MSMARCO small training dataset, which consists of almost 40M tuples of query, a relevant document, and an irrelevant document. For CAR, the model is initialized from pre-trained BERT-base model and trained on 50M samples curated by Nogueira and Cho (2019).

4.3 SDR Configuration and Training

We trained autoencoder variants on a random subset of 500k documents to reduce training time. We incorporate the quantization overhead into the computation of the *compression ratios*, including metadata and the overhead of padding (cf. Appendix A).

In the following sections, we denote the SDR variants as "AESI-{c}-{B}b" where {c} is replaced with the width of the encoded vector and {B} is replaced with the number of bits in the quantization scheme. When discussing AESI with no quantization, we simply write "AESI-{c}".

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

MSMARCO	**Distil bert**	**Late+ AE-24**	**AESI- 16-6b**	**Distilbert +toks**	**AESI-16 -6b+toks**
latency (s)	2.424	1.221	1.106	2.234	1.049
- retrieval	0.106	0.708	0.419	0.126	0.461
- ranking	2.318	0.513	0.687	2.108	0.588
index size (GB)	5.8	90.2	28.6	9.0	31.9
MRR@10	0.390	0.375	0.375	0.390	0.375
CAR	**BERT**	**Late+ AE-24**	**AESI- 16-6b**	**BERT +toks**	**AESI-16 -6b+toks**
latency (s)	6.537	1.847	1.750	6.242	1.720
- retrieval	0.390	0.957	0.650	0.492	0.723
- ranking	6.146	0.891	1.100	5.750	0.997
index size (GB)	26.6	372.1	106.9	38.1	119.0
MAP@1K	0.337	0.189	0.312	0.337	0.312

Table 1: End to end latency comparison with SDR

4.4 End to end Latency Measurement

To measure end to end latency, we configured an OpenSearch[5] cluster in AWS. We used default "production" configurations, with 3 r6g.large datanode machines; disk space was set to 0.5TB. For ranking, we used a single g4dn.xlarge machine, featuring a single T4 GPU instance. This makes the cost of these two components similar.

5 Evaluation Results

In this section, we present the end to end latency results (§ 5.1), show compression ratios and quality tradeoff of the SDR scheme (§ 5.2). We then examine how the proposed autoencoder (§ 5.3) compares with other baselines and present additional measurements (§ 5.4).

5.1 End to End Latency Evaluation

Table 1 (top) shows the latency benefits of SDR on the MSMARCO dataset, assuming document embeddings are stored in the OpenSearch retrieval system and 1k documents are retrieved per query. The Distilbert model (full interaction architecture) has the highest quality and smallest index size (since it is only executed online). However, ranking latency is prohibitively expensive. As a baseline, we use a late interaction model, a two-layer autoencoder with code dimension 24 and float16 quantization, denoted Late+AE-24. For this baseline, the ranking latency is significantly reduced at a cost in terms of quality. However, the document representation

[5] https://aws.amazon.com/opensearch-service/. OpenSearch is a successor to Elasticsearch and based on Lucene.

is large, causing retrieval and overall latency to increase to 0.7 and 1.22 seconds, respectively. SDR, with a dimension of 16 and 6-bits quantization, reaches the same quality as the baseline while striking a better balance between retrieval and ranking latency, reaching overall latency of 1.1 seconds. The index size is also significantly reduced compared to the baseline compression algorithm.

We also consider variants of the algorithms where the documents are pre-tokenized, and the tokenization output is retrieved instead of computing at runtime (marked as +tok in the table). This further improves the ranking latency at the expense of a slight increase in index size. Note that the baseline does not use the raw text and therefore does not benefit from precomputed tokens.

Table 1 (bottom) shows the latency results on the CAR dataset. Here too, the BERT baseline has the highest ranking quality, at the cost of prohibitive latency. The late interaction variants we consider have the same configuration as in the MSMARCO case, where the baseline uses 24 features (with float16 quantization) and SDR uses 16 features (with 6 bits EDEN quantization). Unlike in the MSMARCO case, the quality (i.e., MAP@1k score) of these two options is not similar. This makes SDR better than the baseline in latency, index size, as well as quality (by a large margin of over 14%).

In Appendix D we explore additional configurations and show that the baseline with 52 features reaches the same quality as SDR-16-6b. However, we do not measure end-to-end latency for this case due to the excessive storage size and indexing time. Note that using 52 features for the baseline is expected to have a negative impact on retrieval latency, making the benefits of SDR even more pronounced.

5.2 Compression Rate and Quality Tradeoff

Table 2 shows the results on the MSMARCO query sets for SDR and its compression ratio against storing contextual token embeddings uncompressed. In terms of compression ratio, it can be seen that AESI allows us to massively reduce storage requirements both with and without quantization.

AESI-16-6b reduces storage requirements by 121x, while at the same time showing no significant ranking performance drop. Using AESI-16-6b, a document's embedding can be stored with only 947 bytes and the entire MSMARCO collection can be stored within 8.6GB. There are several advantages of fitting the entire collection's representation into the main memory of the hosting machine, allowing for fast access, further fine-tuning, etc. If further compression rates are required, AESI-8-5b uses just 5 bytes per token, reaching a compression rate of 277x and 487 bytes per document on average. At this level of compression, the entire MSMARCO corpus fits in 3.8GB. The MRR@10 drop is noticeable (0.0119) but still quite low. Finally, for TREC19-DL, the impact of compressing token embeddings is less evident. Only in the most extreme cases such as AESI-4-4b we see a significant drop in nDCG@10 performance. These results demonstrate that the performance drop is very small, showing the effectiveness of our method.

Quant. bits (B)	AESI dim. (c)	Comp. ratio (CR)	MSMARCO-DEV MRR@10	TREC19-DL nDCG@10
32 (float)	16	24	0.3759 (-0.0009)†	0.772 (-0.002)
	12	32	0.3725 (-0.0043)*†	0.784 (+0.01)
	8	48	0.3711 (-0.0057)*†	0.781 (+0.007)
	4	96	0.3660 (-0.0108)*	0.775 (+0.001)
6	16	121	0.3753 (-0.0015)†	0.772 (-0.002)
	12	159	0.3728 (-0.004)*†	0.780 (+0.006)
	8	231	0.3689 (-0.0079)*†	0.775 (+0.001)
	4	423	0.3624 (-0.0144)*	0.766 (-0.008)
5	16	145	0.3735 (-0.0033)*†	0.772 (-0.002)
	12	190	0.3714 (-0.0054)*†	0.778 (+0.004)
	8	277	0.3649 (-0.0119)*	0.770 (-0.004)
	4	506	0.3540 (-0.0228)*	0.767 (-0.007)
4	16	181	0.3665 (-0.0103)*	0.766 (-0.008)
	12	236	0.3639 (-0.0129)*	0.764 (-0.01)
	8	344	0.3544 (-0.0224)*	0.765 (-0.009)
	4	629	0.3408 (-0.036)*	0.752 (-0.022)*
$\text{BERT}_{\text{SPLIT}}$ (Baseline)		1	0.3768	0.774
BM25 (No re-ranking)		1	0.194	0.689

Table 2: *SDR performance in various configurations*: MRR@10 and nDCG@10 are measured over MSMARCO, as described in § 4.1. The absolute difference w.r.t. the $\text{BERT}_{\text{SPLIT}}$ baseline is shown in parentheses. We measured statistical significance in two ways: (1) non-inferiority test with a margin of 0.02, denoted by † when $p < 0.05$, implying that the method is non-inferior to $\text{BERT}_{\text{SPLIT}}$; (2) standard t-test, denoted by * when $p < 0.05$, implying that the difference is statistically significant. The compression ratios indicate the reduction in storage size, including padding and normalization overheads.

5.3 Autoencoder Evaluation

To better understand the impact of the autoencoder, we present MRR@10 results as a function of autoencoder dimensions (i.e., number of floats stored per token) and with the different autoencoder configurations. In addition to the 2-layer AESI architecture we described in § 3.1 (**AESI-2L**), we consider the following variations:

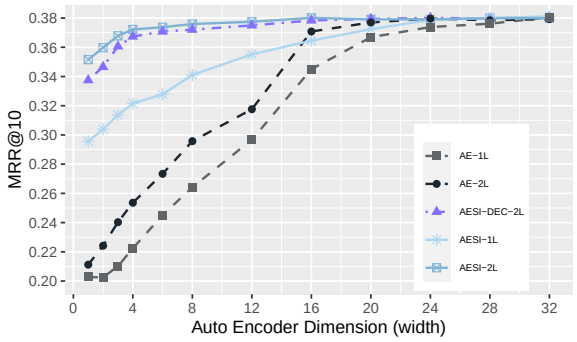

Figure 3: MRR@10 was measured on the MSMARCO-DEV-25 dataset as a function of autoencoder dimensions. The results are shown for standard autoencoders (AE) and our approach (AESI), with single or two-layer encoder and decoder networks. The x–axis shows the dimension of the encoded vector c.

AutoEncoder with 2 Layers (AE-2L). Standard 2-layer autoencoder with gelu activation. This is the same as AESI, only without the side information.

AutoEncoder with 1 Layer (AE-1L). Standard autoencoder with a single dense layer in the encoder and decoder.

AESI with 1 Layer (AESI-1L). AESI with a single dense encoder and decoder layer.

DECoder-only AESI (AESI-DEC-2L). Provides side information to the decoder but not the encoder.

To reduce measurement overhead, we ran the experiment only over the MSMARCO dataset. In addition, we took only the top 25 $\text{BERT}_{\text{SPLIT}}$ passages for each query, denoted MSMARCO-DEV-25, which has a negligible impact on the results. Figure 3 shows the results for the different autoencoder configurations. Providing the side information to the autoencoder proves to be very effective in reducing storage costs, especially when the encoded vector size is small. A 2-layer encoder/decoder model, as expected, is more effective than a single-layer model. The gap is especially large when using side information, showing that the interaction between the encoded vector and the static token embeddings is highly nonlinear. Finally, providing the static embeddings only to the decoder is slightly inferior to providing it also to the encoder.

5.4 Additional Measurements

Quantization Techniques we compare the quantization technique we use to several other techniques, including Deterministic Rounding (Gersho and Gray, 1992), Stochastic Rounding (Connolly et al., 2021), and Subtractive Dithering (Roberts, 1962; Gray and Stockham, 1993). Due to lack of space, the results appear in Appendix B. We found that a randomized Hadamard transform improves quality (assuming similar bit rate), especially in the low-bits regime. Using a quantization technique fitted to the Gaussian distribution of post randomized Hadamard transform data further improve quality, making the EDEN quantization superior to other quantization techniques in our case.

Our scheme uses a fixed number of bits per coordinate, which is essential for performance. However, variable-rate compression can further reduce storage. We used rate-distortion theory (from the information theory field) to upper bound the benefits of such techniques by 11%, which does not seem to justify the added system complexity (cf. Appendix B).

Intrinsic Evaluation of AESI-Encoded Vectors To better understand the impact of side information, we measure the error rate between an input vector and its reconstructed vector (i.e., after encoding and decoding). As expected, in practically all cases, adding the side information reduces error rate compared to a 2-layer autoencoder (AE-2L) with the same code dimension.

In IR, the document frequency of a token is known to be negatively correlated with the token's importance. We found that the error rate for AE-2L decreases with frequency, while the error rate for AESI *increases* with frequency. This shows that the AESI scheme can better focus on tokens that are important for ranking. A possible explanation for this phenomena is that the static embeddings for infrequent tokens are more informative (i.e., more helpful as side information) compared to static embeddings for frequent tokens (e.g., 'the'). We also found AESI excels more in compressing nouns, verbs, and adjectives, while AE-2L excels more in compressing punctuation, determiners, and adpositions. Again, this demonstrate that the static embeddings is most helpful in encoding tokens that are crucial for ranking. The details of this evaluation are provided in Appendix C.

6 Conclusion

In this paper, we proposed a system called SDR to solve the storage cost and latency overhead of existing late-interaction transformer based models for passage re-ranking. The SDR scheme uses a novel

autoencoder architecture that uses static token embeddings as side information to improve encoding quality. In addition, we explored different quantization techniques and showed that the recently proposed EDEN performs well in our use case and presented extensive experimentation. Overall, the SDR scheme reduces pre-computed document representation size by 4x–11.6x compared to a baseline that uses existing approaches.

In future work, we plan to continue investigating means to reduce pre-computed document representation size.We believe that additional analysis of BERT's vector and their interaction with the context would be fundamental in such an advancement.

References

Nir Ailon and Bernard Chazelle. 2006. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC, page 557–563, New York, NY, USA. Association for Computing Machinery.

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-Efficient Sgd via Gradient Quantization and Encoding. *Advances in Neural Information Processing Systems*, 30:1709–1720.

RCM Barnes, EH Cooke-Yarborough, and DGA Thomas. 1951. An electronic digital computor using cold cathode counting tubes for storage. *Electronic Engineering*.

Qingqing Cao, Harsh Trivedi, Aruna Balasubramanian, and Niranjan Balasubramanian. 2020. DeFormer: Decomposing pre-trained transformers for faster question answering. In *ACL*, pages 4487–4497, Online. Association for Computational Linguistics.

Jiecao Chen, Liu Yang, Karthik Raman, Michael Bendersky, Jung-Jung Yeh, Yun Zhou, Marc Najork, Danyang Cai, and Ehsan Emadzadeh. 2020. DiPair: Fast and accurate distillation for trillion-scale text matching and pair modeling. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2925–2937, Online. Association for Computational Linguistics.

Michael P. Connolly, Nicholas J. Higham, and Theo Mary. 2021. Stochastic rounding and its probabilistic backward error analysis. *SIAM Journal on Scientific Computing*, 43(1):A566–A585.

Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA.

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. In *TREC*.

Mohaned Essam, Tong Boon Tang, Eric Tatt Wei Ho, and Hsin Chen. 2017. Dynamic point stochastic rounding algorithm for limited precision arithmetic in deep belief network training. In *2017 8th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 629–632.

Bernard J. Fino and V. Ralph Algazi. 1976. Unified matrix treatment of the fast walsh-hadamard transform. *IEEE Transactions on Computers*, 25(11):1142–1146.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020a. Earl: Speedup transformer-based rankers with pre-computed representation. *arXiv: Information Retrieval*.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020b. Modularized transfomer-based ranking framework. In *EMNLP*, pages 4180–4190.

A. Gersho and R. M. Gray. 1992. *Vector quantization and signal compression*. Kluwer Academic Publishers.

R.M. Gray and T.G. Stockham. 1993. Dithered quantizers. *IEEE Transactions on Information Theory*, 39(3):805–812.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *ICML*.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *ICML*, volume 37 of *Proceedings of Machine Learning Research*, pages 1737–1746, Lille, France. PMLR.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *ICML*, volume 1, pages 3929–3938.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020a. Improving efficient neural ranking models with

cross-architecture knowledge distillation. *arXiv preprint arXiv:2010.02666*.

Sebastian Hofstätter, Markus Zlabinger, and Allan Hanbury. 2020b. Interpretable & time-budget-constrained contextualization for re-ranking. In *ECAI*, pages 513–520.

Kathy J Horadam. 2012. *Hadamard Matrices and Their Applications*. Princeton university press.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

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 *EMNLP*, pages 6769–6781.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *SIGIR*, pages 39–48.

Jakub Konečnỳ and Peter Richtárik. 2018. Randomized Distributed Mean Estimation: Accuracy vs. Communication. *Frontiers in Applied Mathematics and Statistics*, 4:62.

Wenhao Lu, Jian Jiao, and Ruofei Zhang. 2020. Twinbert: Distilling knowledge to twin-structured bert models for efficient retrieval. *arXiv preprint arXiv:2002.06275*.

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 *SIGIR*, pages 49–58.

Avner May, Jian Zhang, Tri Dao, and Christopher Ré. 2019. *On the Downstream Performance of Compressed Word Embeddings*. Curran Associates Inc., Red Hook, NY, USA.

Ilan Newman. 1991. Private vs. Common Random Bits in Communication Complexity. *Information processing letters*, 39(2):67–71.

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 *CoCo@NIPS*.

Ping Nie, Yuyu Zhang, Xiubo Geng, Arun Ramamurthy, Le Song, and Daxin Jiang. 2020. Dc-bert: Decoupling question and document for efficient contextual encoding. In *SIGIR*, pages 1829–1832.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *CoRR*, abs/1901.04085.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *arXiv: Information Retrieval*.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2020. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2010.08191*.

L. Roberts. 1962. Picture coding using pseudo-random noise. *IRE Transactions on Information Theory*, 8(2):145–154.

Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc.

Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. 2017. Distributed Mean Estimation With Limited Communication. In *International Conference on Machine Learning*, pages 3329–3337. PMLR.

Shay Vargaftik, Ran Ben Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben-Itzhak, and Michael Mitzenmacher. 2022. Eden: Communication-efficient and robust distributed mean estimation for federated learning.

Shay Vargaftik, Ran Ben-Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben-Itzhak, and Michael Mitzenmacher. 2021. Drive: One-bit distributed mean estimation. *Advances in Neural Information Processing Systems*, 34.

Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. 2018. Training deep neural networks with 8-bit floating point numbers. In *NIPS*, pages 7686–7695.

Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. 2018. Training and inference with integers in deep neural networks. In *ICLR*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwikj. 2021. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *ICLR*.

Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the use of lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1253–1256, New York, NY, USA. Association for Computing Machinery.

Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. 2021. Pretrained transformers for text ranking: BERT and beyond. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Tutorials*, pages 1–4, Online. Association for Computational Linguistics.

A EDEN Quantization

In this Appendix, we include an overview of the quantization method we adapted to our use case. The algorithm is summarized in Algorithm 1, for full details see Vargaftik et al., 2022, Section 3.

We start by introducing the following definitions:

Definition 1 (Horadam, 2012)**.** *A normalized Walsh-Hadamard matrix,* $H_{2^k} \in \{+1,-1\}^{2^k \times 2^k}$*, is recursively defined as*

$$H_1 = 1; \;\; H_{2^k} = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{pmatrix}.$$

Definition 2 (Ailon and Chazelle, 2006)**.** *A randomized Hadamard transform,* $\mathcal{H}$*, of a vector,* $x \in \mathbb{R}^{2^k}$*, is defined as* $\mathcal{H}(x) \coloneqq H_{2^k} D x$*, where* H_{2^k} *is a normazlized Walsh-Hadmard matrix, and* D *is a diagonal matrix whose diagonal entries are i.i.d. Rademacher random variables (i.e., taken uniformly from* $\{+1,-1\}$*). While* $\mathcal{H}$ *is randomized and thus defines a distribution, when* D *is known, we abuse the notation and define the inverse Hadamard transform as* $\mathcal{H}^{-1}(x) \coloneqq (H_{2^k} D)^{-1} x = D H_{2^k} x$*.*

The quantization operates as follows. Given a vector, denoted $x \in \mathbb{R}^d$, we first *precondition* it using a randomized Hadamard transform, $\mathcal{H}$, and normalize by multiplying by $\sqrt{d}/\|x\|_2$. There are several desired outcomes of this transform[6]. First, the dynamic range of the values is reduced (measured, for instance, by the ratio of the ℓ_∞ and the ℓ_2 norms). Loosely speaking, we can think of the transform as spreading the vector's information evenly among its coordinates. Second, regardless of the distribution of the input vector, each coordinate of the transformed vector will have a distribution that is close to the standard Gaussian distribution (as an outcome of the central limit theorem). After the transform, we perform scalar quantization that is optimized for the $\mathcal{N}(0,1)$ distribution, using K-means (also known as Max-Lloyd in the quantization literature (Gersho and Gray, 1992)), with $K = 2^B$. The vector X of cluster assignments together with the original vector's ℓ_2 norm can now be stored as the compressed representation of the original vector.

To retrieve an estimate of the original vector, we perform the same steps in reverse. We replace the vector of cluster assignments X with a vector $\hat{y}$ containing each assigned cluster's centroid, denormalize, and then apply the inverse randomized Hadamard transform, $\mathcal{H}^{-1}$. To avoid encoding D directly, we recreate it using *shared randomness* (Newman, 1991) (e.g., a shared pseudorandom number generator seeded from a hash of the vector's text).

[6] We also note that the transform has the advantage of having a vectorized, in-place, $O(d \log d)$-time implementation (Fino and Algazi, 1976).

Algorithm 1 B-bits Vector Quantization (EDEN) (Vargaftik et al., 2022, Section 3)

$\mathcal{H}$ - A randomized Hadamard transform
$\mathbb{C}$ - K-Means centroids over the normal distribution, where $K = 2^B$

Quantize($x \in \mathbb{R}^d$)**:**
$y := \frac{\sqrt{d}}{\|x\|_2} \mathcal{H}(x)$
Compute $X \in \{0, \ldots, 2^B - 1\}^d$
s.t. $X_i = \arg\min_k |y_i - \mathbb{C}_k|$
return X, $\|x\|_2$

Dequantize(X, $\|x\|_2$)**:**
Compute $\hat{y} \in \{\mathbb{C}_0, \ldots, \mathbb{C}_{2^B-1}\}^d$ s.t. $\hat{y}_i = \mathbb{C}_{X_i}$
return $\hat{x} = \mathcal{H}^{-1}\left(\frac{\|x\|_2}{\sqrt{d}} \hat{y}\right)$

Block-wise Quantization. The AESI encoder reduces the dimension of the contextual embeddings from hundreds (e.g., 384) to a much smaller number (e.g., 12). On the other hand, the randomized Hadamard transform's preconditioning effect works best in higher dimensions (Ailon and Chazelle, 2006). In order to resolve this conflict, we first concatenate the reduced-dimension vectors of all the tokens from a single document. We then apply the Hadamard transform with a larger block size (e.g., 128) on the concatenated vector, block-by-block (padding the last block with zeros when necessary). When evaluating the compression efficiency, we consider the overhead incurred from (a) the need to store the vectors' ℓ_2 norms and (b) the padding of the final Hadamard block in a concatenated vector. Balancing these factors should be done per use case. We empirically measured the padding overhead (for a block size of 128) for AESI 4, 8, 12, and 16 to be 20.1%, 9.7%, 6.7%, and 4.5% for the MSMARCO dataset and 41%, 18.5%, 12.5%, and 8.4% for the CAR dataset.

B Quantization Evaluation

To study the impact of quantization, we fix AESI-16 as our baseline and measure how different quantization strategies and number of bits affect the MRR@10 score. Note that we do not measure quantization over the baseline $\text{BERT}_{\text{SPLIT}}$ since it can only achieve a compression ratio of up to 32x per coordinate (using 1 bit per coordinate). In addition to **EDEN** (Appendix A, Algorithm 1), we consider the following quantization strategies:

Deterministic Rounding (DR) (Gersho and Gray, 1992). Maps the input coordinates into the $[0, 2^B - 1]$ range using min-max normalization and rounds to the nearest integer.

Stochastic Rounding (SR) (Barnes et al., 1951; Connolly et al., 2021). Normalizes as before using min-max normalization, and additionally adds a uniform *dither* noise in $(-0.5, 0.5)$ and then rounds to the nearest integer. This provides an unbiased estimate of each coordinate.

Subtractive Dithering (SD) (Roberts, 1962; Gray and Stockham, 1993). Same as SR, only now before denormalization, instead of just using the values in $\{0, \ldots, 2^B - 1\}$, we first subtract the original dither noise, which we assume can be regenerated using shared randomness. This is an unbiased estimator with reduced variance.

Hadamard Variants (H-DR, H-SR, and H-SD). These variants correspond to the previous methods; only they are preceded by a randomized Hadamard transform.

EDEN with Bias Correction (EDEN-BC) (Vargaftik et al., 2022, Section 2.3). This variant of EDEN optimizes for lower bias over the mean squared error (MSE) by multiplying the dequantization result in Algorithm 1 by a bias correction scalar: $\|x\|_2^2 \,/\, \langle \mathcal{H}(x), \hat{y} \rangle$.

Figure 4 shows the results for the different quantization methods. First, we observe that the Hadamard variants perform better than their non-Hadamard counterparts. Second, we see that EDEN performs better than all other schemes. The differences are more pronounced in the low-bit regime, where the choice of quantization scheme has a drastic impact on quality. We also note that unlike in other use cases, such as distributed mean estimation, bias correction is inappropriate here and should not be performed at the cost of increased mean squared error (MSE). This conclusion follows by observing that EDEN and the deterministic rounding methods (DR, H-DR) are respectively better than EDEN-BC and the stochastic rounding methods (SR, H-SR). We add that the subtractive dithering methods (SD, H-SD), expectedly, work the same or better than their deterministic counterparts since they produce a similar MSE while also being unbiased.

The current quantization scheme requires padding to full 128 blocks. For AESI with a small code size, the padding overhead may reach 10% – 20% percent. In addition, we send a normalization value per 128-block, which we currently send as a float32 value, adding 4% – 5% additional overhead. Padding can be reduced by treating the last 128-block separately, e.g., applying a method that does not require Hadamard transform. Normalization overhead can be reduced, e.g., by sending normalization factors as float16 instead of full float32. However, such solutions complicate the implementation while providing limited storage benefits, hence, they were not explored in the context of this paper.

Beyond Scalar Quantization. Scalar quantization using a fixed number of bits is a suboptimal technique in general since it does not allocate fewer bits for more frequent cases. Entropy coding(Gersho and Gray, 1992) can do better in this aspect. However, this improvement seems may not justify the added complexity: For the case of 6-bit quantization, the entropy of the quantization indices turned out to be 5.71bit, indicating that the compression gain is limited to about 5% in this case (even before accounting for the overhead incurred from the entropy coding algorithm itself). Additional directions include quantizing multiple values together (vector quantization), as well as designing the quantizer with entropy consideration in mind (entropy-constrained vector quantization). In order to estimate the potential gains of all these methods combined, we turn to information theory, and rate-distortion theory in particular, which studies the optimal tradeoffs between distortion and compression rate (Cover and Thomas, 2006). For a Gaussian source, which is a reasonable approximation of the vectors that are compressed in our case (following the randomized Hadamard transform), it is known that the optimal (lossy) compression rate is given by $\frac{1}{2}\log_2(\frac{1}{MSE})$, where MSE is the mean squared error of the compressed source. We computed the optimal rate that is achievable for the

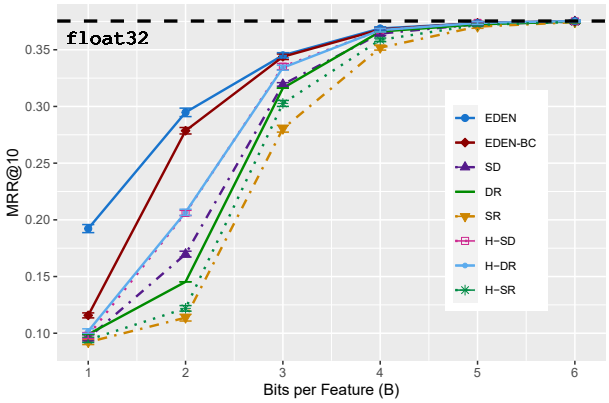

Figure 4: MRR@10 for different quantization methods. Each run quantizes and dequantizes AESI-16 encoded documents over the MSMARCO-DEV-25 dataset. For each randomized quantization method and number of bits, we take the average of 10 runs (the error bars show the standard deviation).

MSE that our system achieved for 6 bits, and the optimal rate was 5.35bit, indicating a potential gain of 11%. Given these results, and also given that for other bit rates the results were similar, we conclude that further quantization improvements have limited gain, which most likely does not justify the added system complexity.

C Intrinsic Evaluation of AESI-Encoded Vectors

In the body of the paper, we showed the effectiveness in ranking and utility in compression rates of AESI over AE architectures. However, such evaluations do not capture the encoded information at the token-level. In this intrinsic evaluation we try to discern when and why adding the static embedding as side information contributes to better capturing the token meaning.

We study the effectiveness of different autoencoder configurations in reconstructing back the original token vector, as measured through the MSE between the original vector and the reconstructed vector:

$$MSE\left(v, D\left(E(v,u),u\right)\right) \ ,$$

where v is a contextualized vector ($\text{BERT}_{\text{SPLIT}}$ output at layer 10), u is the static embedding, and the encoder $E(v,u)$ and the decoder $D(e,u)$ are as defined in § 3.1. High MSE scores indicate the inability of the autoencoder to encode the original vector's information.

Document Frequency: One way to assess the importance of a document w.r.t. a query is through the inverse document frequency of query tokens, typically measured through TF-IDF or BM25 schemes (Robertson and Zaragoza, 2009). In principle, the more infrequent a query token is in a document collection, the higher the ranking of a document containing that token will be. Tokens with (very) high frequencies are typically stop words or punctuation symbols, which have lower importance when determining the query-document relevance.

Based on this premise, we study how MSE varies across token frequency. We selected a random sample of 256k documents from MSMARCO, tokenized them, and run them through $\text{BERT}_{\text{SPLIT}}$ to get 20M contextualized token representations. Then, for each token we measured their document frequency as $DF(t) = \log_{10}(|\{d \in D : t \in d\}|/|D|)$ (where D is our document collection), and in Figure 5 we plot the average MSE against the rounded DF scores. From this experiment, we make the following observations.

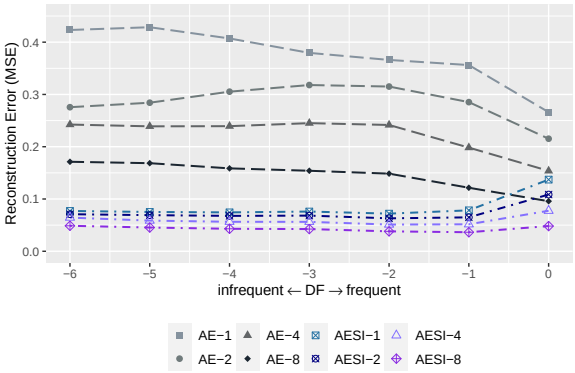

Figure 5: Reconstruction Error vs. DF for the different AE and AESI configurations. AESI shows robust performance in recovering back the token's representation with a MSE score (y-axis), which is constant for documents with varying DF scores. It is interesting to note that for frequent tokens (i.e., tokens that are function words, hence play a marginal role in retrieval), the error rate is higher when compared to the rest of the tokens.

First, on all encoded width configurations, our approach, AESI, consistently achieves lower MSE compared to the AE architecture (for all DF values). Lower MSE correlates to a better ranking quality, as shown in § 5.3. Furthermore, for tokens with low DF, adding the static side information during the training of AESI for compression provides a huge advantage, which shrinks when the token is present in many documents in the collection.

Second, on the end spectrum of high-frequency tokens, we note a downwards trend for AE and

an upwards trend for AESI, especially for $DF \in [-1, 0]$. The MSE decrease for AE is expected since the training data contains more frequent tokens. The increase for AESI can be explained given that in this frequency range, we deal with tokens that are function words (e.g., 'the') whose role is more in tying up content within a sentence and has less standalone meaning. In this case, static embeddings cannot capture context, which reduces the contribution provided by the side information.

D Compression Results on TREC CAR

In Table 3 we show MAP@1K results on the TREC CAR dataset for a varying number of features. We compare the baseline – an autoencoder with 2 layers and float16 quantization – to SDR scheme, with the same number of features and EDEN 6bits quantization. The SDR scheme is able to provide solid results even for 3 features with a MAP@1K score of 0.268. With the baseline method, similar results are achieved only with 36 features. As a comparison, this is much higher than BM25 using the Answerini system (Yang et al., 2017), which reaches 0.156 MAP@1K score. With 16 features (the configuration we used for Table 1), SDR reaches a score of 0.311, compared to 0.312 for the baseline with 52 features. Finally, with the largest size of features we tested, 64, the baseline reached a MAP@1K score of 0.313, similar to the score achieved by SDR-20 (0.314), demonstrating the effectiveness of the static embeddings as side information.

E Late Interaction Model Illustration

In Figure 6 we illustrate the architecture of the late interaction model vs. the standard BERT model. In standard BERT, the query and documents are concatenated before the first BERT layer. Therefore, if K documents are ranked, all BERT transformers layers are applied K times. In the late interaction architecture, the bottom L layers (e.g., 10 transformer layers shown in the figure) are executed independently for the query and the documents. The document representation is precomputed and stored in the index. During online execution, the query representation is computed once, and the document representations are retrieved from the index. Only the interaction block (e.g., 2 transformer layers) are executed K times, once for each ranked document.

Num features	AE-2L-float16	AESI-{X}-6b
1	0.0617*	0.1993*
2	0.0411*	0.2064*
3	0.0479*	0.2682*
4	0.0448*	0.2790*
6	0.0461*	0.2968*
8	0.0573*	0.2987*
12	0.0795*	0.3106*
16	0.1197*	0.3116*
20	0.1530*	0.3146*
24	0.1889*	0.3166
28	0.2317*	0.3162*
32	0.2464*	0.3188
36	0.2668*	0.3195†
40	0.2877*	0.3205†
44	0.2967*	0.3206†
48	0.3018*	0.3211†
52	0.3127*	0.3212†
56	0.3135*	0.3212†
60	0.3156*	0.3202†
64	0.3137*	0.3218†
$\text{BERT}_{\text{SPLIT}}$		0.3217

Table 3: Comparing SDR and the baseline (2 layer autoencoder with float16 quantization) on TREC CAR. We present MAP@1K score for a varying number of code size between 1 and 64.

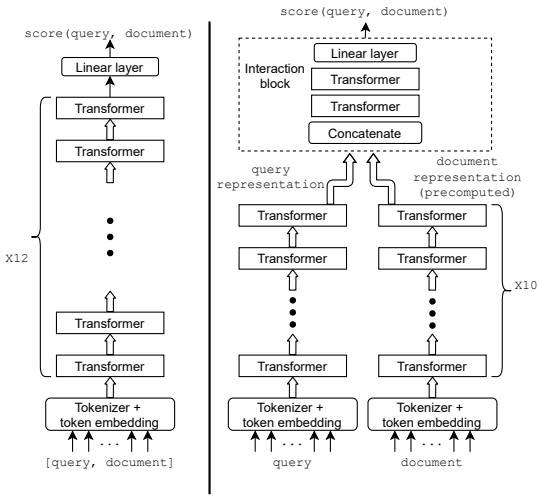

Figure 6: Left: BERT ranker. Right: late-interaction ranker (with two transformer layers as the interaction block).