OySTER: ORTHO-SPHERICAL TRANSFORMATION TO PRESERVE SEMANTICS IN EMBEDDING COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

The impetus towards larger language models, stemming from better world knowledge along with enhanced semantic and contextual understanding, has led to diverse text and content representation via vector embeddings of *extremely high dimensionality*. However, this poses severe challenges in terms of embedding storage (in vector databases) and computational inefficiencies (for similarity and nearest neighbors) in downstream applications like search and retrieval.

This paper presents OySTER, a novel embedding compression method combining *orthogonal eigen-space projection* of high dimensional text embedding (to align with informative dimensions) along with *truncated hyper-spherical coordinate transformation*. Extensive experiments on *MTEB* benchmark datasets depict improved performance of our approach, in terms of semantic understanding and retrieval tasks, even under $64\times$ compression ratio. Additionally, we show that OySTER inherently provides *Matryoshka embeddings* as a by-product – performing better than the models trained with Matryoshka learning objective.

1 INTRODUCTION & RELATED WORK

Large Language Models (LLMs) have become ubiquitous across diverse modern-day applications in Artificial Intelligence (AI), Computer Vision (CV), and Natural Language Processing (NLP) to name a few. The encapsulation of world knowledge with semantic and contextual understanding of contents represented by vector embeddings (from LLMs) powers frameworks like Retrieval Augmented Generation (RAG), agentic search, classification, similarity computation, and multimodal retrieval. To this end, larger and larger models (trained with enormous data from diverse corpora) have depicted near human-level performance in terms of content understanding, classification, and search tasks, as presented via several open-source benchmark dataset evaluations.

Motivation. Such large embedding models provide an encoding of contents in terms of extremely *high-dimensional dense real-valued vectors*. As such, if a typical model produces 1024-dimensional *float32* vector representation, a single embedding would require 4 KB of storage. At this scale, storage of embedding representations for only 100 million data points would require a whooping 400 GB of space, practically rendering it computationally infeasible for most large-scale application. In fact, state-of-the-art LLMs generate vector representations with even larger dimensionality – for instance, the embedding dimensionality of Qwen3-8B, DeepSeek-V3 and LLaMA-3-70B models ranges from 4096 to 8192, while the latest GPT-6 Astra uses a $16,384$-dimensional space!

The problem is exacerbated in scenarios involving multi-vector approaches, like retrieval and search tasks using late-interaction paradigm (e.g., ColBERT Khattab & Zaharia (2020)) with individual token embeddings (instead of document representation) – effectively increasing the storage requirements by a factor of tens or hundreds. This poses severe challenges not only in terms of embedding storage (in vector databases), but also results in computational inefficiencies (for similarity and nearest neighbor search) in downstream applications. Further, the *curse of dimensionality* tends to affect similarity and approximate neighborhood search tasks. Thus, embedding dimensionality reduction or compression constitutes a necessary domain of research in the current LLM dominated landscape.

Note, embedding compression (the central theme of this work) differs significantly from the that of model compression, which focuses on compressing the layer weights and modules of a language

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

model itself rather than the embedding dimension it generates. Embedding compression finds application in NLP, CV Babenko & Lempitsky (2024), and recommendations Kasalický et al. (2025).

Related Work. Traditional algorithms for *lossless compression* such as Lempel-Ziv-Welch (LZW) Welch (1984) and zstd, forming the backbone of compressed formats like png, gif and zip, provide storage efficiency without any information or performance loss. However, since semantic relevance or similarity is usually computed via the *cosine similarity* measure, such compressed representation needs to be decompressed repeatedly (for example, in KV-cache) for computing the vector similarity. Further, the vector dot-product (for cosine similarity) is performed in high-dimensional space (after decompression), leading to high latency and computational inefficiencies.

For real-valued vector embeddings, state-of-the-art *lossless* approaches tend to transpose the embedding matrix, byte-shuffle to group exponent bytes, and then apply entropy coding Hershcovitch et al. (2025) to achieves around $1.2\times$ compression. Recently, conversion of Cartesian vectors to equivalent spherical coordinates before byte shuffling and entropy coding has been shown to achieve $1.5\times$ compression. It leverages the fact that in spherical transform the exponents and high-order mantissa bits of float format are concentrated around a fixed value Xiao (2026). However, lossless compression techniques do not provide a viable reduction in storage requirements, at large scale.

As such, the vast majority of work in embedding compression have considered *lossy techniques*. A common approach is *quantization*, which converts the embeddings from high-precision format (float32 or FP32) to lower-precision formats (like BF16, FP16 or Int8) Burgess et al. (2019) for storage. However, reducing the precision introduces *quantization noise*, which degrades the similarity computation accuracy. Further, reduction of precision can only be performed to pre-defined levels, along with high similarity computation latency (still computed on high dimensionality).

Another line of related work involves the *product quantization* (PQ) technique Jégou et al. (2011), wherein the embedding vectors are partitioned into subspaces (based on a subset of the dimensions) and clustered, to represent each subspace using the closest centroid index. Different variants like Optimized PQ (OPQ) Ge et al. (2014) and ScaNN Guo et al. (2020) have been proposed to improve the accuracy of the compressed representation. However, although PQ provides a highly efficient lossy compression technique, it is specifically designed for fast, large-scale vector similarity search and does not support cosine similarity computation directly on the compressed representations.

Several recent works employ lossy quantization on *polar or spherical coordinates* for compression, for example quantization of the spherical angles to fixed bit-widths was proposed in Trojak & Witherden (2021). Recent works like PolarQuant Wu et al. (2025) and PCDVQ Yue et al. (2025) combine the idea of quantization with spherical coordinate transformation, providing an avenue for memory-accuracy tradeoff with higher latency for cosine similarity computation.

The above methods work on the principle of reducing the precision or modifying the *float32* format for embedding storage, thereby reducing the memory footprint. *Dimensionality reduction*, an alternative approach, aims to transform the high-dimensional vector representations into low-dimensional space, while aiming to preserve the local and global geometry of the embedding space. The ensuing low-dimensional embeddings are stored in the original *float32* format. Along with space efficiency, this approach also provides a number of advantages – (i) no requirement of reconstructing the original embeddings for computing semantic similarity, (ii) computational efficiency of vector dot-product operation with cosine similarity being calculated directly in the low-dimensional space, and (iii) alleviates the curse-of-dimensionality for similarity and nearest neighbor search tasks.

The most common technique used is *Principal Component Analysis* (PCA) which transforms the original high-dimensional vectors into low-dimensions by projecting them onto a subspace of eigen-decomposition. While PCA works on linear relationships between the original dimensions, *kernel-PCA* Schölkopf et al. (1997) provides an extension for non-linearity based dimensionality reduction. On the other hand, *Uniform Manifold Approximation and Projection* (UMAP) McInnes et al. (2018) utilizes graph-based neighborhood and Riemannian geometry for non-linear dimensionality reduction. In a similar spirit, probabilistic distribution based learning is used in *t-Distributed Stochastic Neighbor Embedding* (t-SNE) van der Maaten & Hinton (2008) for non-linear dimensionality reduction. UMAP and t-SNE have been shown to work particularly well for reducing to extremely low dimensions, specifically for visualization. Deep learning approaches like *Auto-Encoders* can also be trained with reconstruction loss based objective to extract low-dimensional latent representations, but have been shown to be similar to PCA in terms of performance.

To enable ease of dimensionality reduction, *Matryoshka embeddings* have been proposed, wherein the embedding model itself is trained using a multi-scale loss function, termed as *Matryoshka Representation Learning* (MRL) Kusupati et al. (2022). This training process aims to keep the most critical and broader semantic concepts in the first few dimensions with finer-grained semantic nuances represented within the later dimensions. Thus, the original high-dimensional embedding vector can be easily sliced or truncated to a lower dimensionality (depending on application requirement) without a complete collapse of semantic knowledge captured within the truncated embeddings.

Contributions: This work tackles the problem of *embedding dimensionality reduction* or embedding compression (used interchangeably in this document) to bring best-of-both-worlds in terms of storage requirements and computational efficiency. We propose the novel **O***rthogonal and Hy*per-**S***pherical* **T***ransform for* **E***mbedding* **R***eduction* (`OySTER`) approach, wherein:

- We couple eigen-decomposition based projection of the original high-dimensional vector embeddings, hyper-spherical coordinate transformation, and truncation operation to obtain low-dimensional representation of contents;
- We showcase that the obtained low-dimensional embedding space captures the overall semantic information more efficiently than other dimensionality reduction techniques even under high compression ratios;
- `OySTER` inherently produces Matryoshka embeddings as a by-product from any language model post-training, removing the need for costly model training procedure – as well as depicting significantly better performance than models trained with MRL objective; and
- Extensive empirical evaluation on diverse semantic understanding tasks from the MTEB benchmark showcases the efficacy of our proposed `OySTER` method.

2 Preliminaries & Intuition

Hyper-Spherical Transformation. Using the hyper-spherical coordinate transformation, a d-dimensional vector $\boldsymbol{u}$ can be losslessly represented by the radius $r = ||\boldsymbol{u}||_2$ and $d-1$ angular values $\theta_1, \theta_2, \cdots, \theta_{d-1}$.

As shown in Figure 1, consider a unit-norm 3-dimensional vector $\boldsymbol{u} = (x, y, z)$ to be represented by 2 spherical coordinates. Thus, $r = ||\boldsymbol{u}||_2 = 1$ since the vector is unit norm, and by geometric properties based on the angles between $\boldsymbol{u}$ and the coordinate axes we have,

$z = \cos\theta, \;\; x = \sin\theta\cos\phi, \;\; \text{and} \;\; y = \sin\theta\sin\phi$

Thus, our vector $\boldsymbol{u}$ can be equivalently represented in Euclidean and Spherical coordinates as $\boldsymbol{u} = (x, y, z) = (\theta, \phi)$. From the above formulation, we can observe, $\frac{y}{x} = \frac{\sin\theta\sin\phi}{\sin\theta\cos\phi} = \tan\phi$. Thus, using trigonometric relations, $\phi = \arctan(\frac{y}{x})$ and $\theta = \arctan(\frac{\sqrt{x^2+y^2}}{z})$.

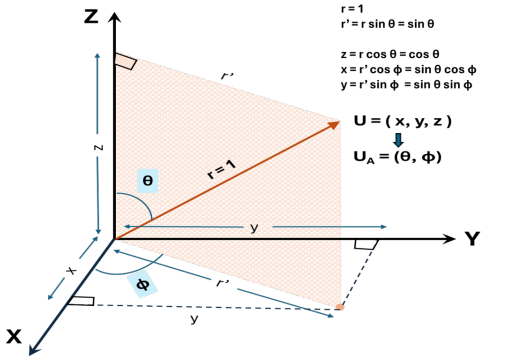


Figure 1: Pictorial depiction of spherical coordinate transformation from Euclidean coordinates for a 3-dimensional vector.

Generalizing the above, the angular values for a unit-norm d-dimensional vector $\boldsymbol{u} = (x_1, x_2, \cdots, x_d)$ can thus be computed efficiently using the *arctan* trigonometric formulation, as:

$$\theta_i = \arctan\left(\frac{\sqrt{x_{i+1}^2 + x_{i+2}^2 + \cdots + x_d^2}}{x_i}\right) \quad \text{for } 1 \le i \le d-2 \quad \text{and, } \theta_{d-1} = \arctan\left(\frac{x_d}{x_{d-1}}\right) \tag{1}$$

Thus, we have $\boldsymbol{u} = (x_1, x_2, \cdots, x_d) = (\theta_1, \theta_2, \cdots, \theta_{d-1})$, with $r = ||\boldsymbol{u}||_2 = 1$.
The lossless reconstruction from the hyper-spherical coordinates back to the Cartesian values can also be efficiently computed using cascading sin and cos operations, as shown below,

$$x_j = \left[\prod_{i=1}^{j-1} \sin\theta_i\right] \cos\theta_j \quad \text{for } 1 \le j \le d-1 \quad \text{and,} \quad x_d = \left[\prod_{i=1}^{d-1} \sin\theta_i\right] \sin\theta_{d-1} \tag{2}$$

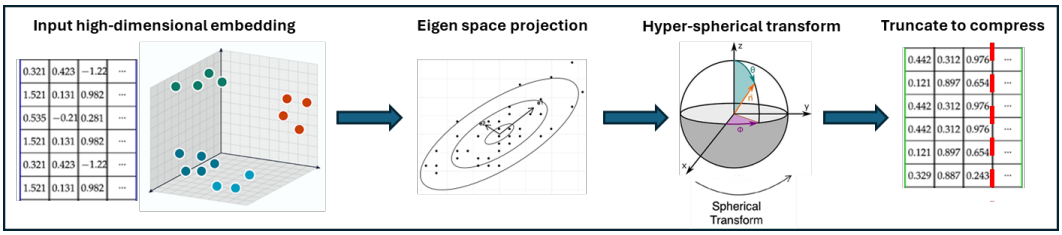


Figure 2: Architectural overview & pipeline of the proposed OySTER embedding compression framework, combining eigen-space projection and truncated hyper-spherical transformation.

Observation 1. *sine values range between* $[-1, 1]$*, and thus* $|sin| \leq 1$*. Hence, a cascading multiplicative operation of* sin *function, would typically generate an extremely small number in terms of magnitude. That is,* $|\prod_{i=1}^{k} \sin\theta_i| \to 0$ *as* $k \to \infty$*. Thus, reconstruction of an extremely high-dimensional vector would render the magnitude of the lower dimensions typically close to* 0*. In terms of vector dot-product, these terms would have negligible contributions, and hence the lower dimensions can be truncated with minimal or near-zero loss of cosine similarity information.*

Observation 2. *The computation of angular coordinates for a particular dimension, depends on all the other preceding Cartesian coordinate values, as seen in the equations above. Thus, in principle, partial global semantic information (across different dimensions) is captured to a certain extend in each of the angular values. Thus, even if some angular values are dropped or truncated, fragments of information from the global embedding space would still reside within the retained angular dimensions – whereas in PCA the information of the discarded eigenspaces is completely lost.*

However, it is key to note that for high-dimensional vectors the angular coordinate values concentrate around a fixed value, as discussed in Xiao (2026). Thus, determination of which angular dimension is important and should be retained is non-trivial. Although, other angular values contain partial information of the truncated dimensions, blind angular coordinate truncation might potentially lead to severe degradation in semantic information present in the compressed embedding space, in the event that the angular value corresponding to an important embedding dimension was discarded.

Singular Value Decomposition (SVD). SVD forms a foundational linear algebra matrix factorization technique, wherein any real-valued matrix $\boldsymbol{A}$ of order $m \times n$ can be factorized into 3 matrices as, $\boldsymbol{A}_{m\times n} = \boldsymbol{U}_{m\times m}\boldsymbol{\Sigma}_{m\times n}\boldsymbol{V}^T_{n\times n}$. Here, $\boldsymbol{U}$ is an orthogonal matrix with its columns representing the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$, $\boldsymbol{\Sigma}$ is diagonal matrix containing the square roots of eigenvalues sorted based on importance or strength, and $\boldsymbol{V}^T$ is another orthogonal matrix with its rows representing the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$. SVD initially identifies the dimension with the maximum variance, and then iteratively finds other orthogonal dimensions that capture the remaining variance. This provides a linear transformation based on rotation and stretch. Since, the singular values in $\boldsymbol{\Sigma}$ are sorted by importance, we can extract the top-k eigenvalues and the corresponding eigenvectors to create a low-rank approximation of the original matrix, which forms the operating principle of PCA.

Observation 3. *The inherent sorting of dimensional importance obtained via eigen-decomposition provides a natural ordering of dimensions that capture the maximum semantic information. Thus, by projecting the original vector embedding onto the eigen-space, we can obtain a transformed ordered embedding representation, wherein the first dimension captures the most salient or important semantic information (and so on) present within a model embedding space.*

3 Overview of the OySTER Approach

Based on the above observations and intuitions, we now present the overview and overall pipeline of our proposed OySTER methodology for embedding compression, as presented in Figure 2.

Preprocessing. Given a large language model $\mathcal{M}$, we initially compute the high-dimensional vector embeddings on a training dataset $\mathcal{T}$. The obtained embedding vector space is then factorized via singular value decomposition (SVD) to obtain the ordered eigenvectors, which are then stored as

the characteristic space of model $\mathcal{M}$. This provides us with the information as to which embedding dimensions of $\mathcal{M}$ constitutes important semantic information.

Embedding Dimension Compression. For compressing an input d-dimensional vector embedding ε obtained from $\mathcal{M}$, OySTER first transforms it by projecting it onto the stored model eigenvectors. This step generates a transformed d-dimensional vector ε', in which the dimensions are ordered in terms of importance of semantic information contained in them (Observation 3). That is, the first dimension of ε' captures the maximum semantic similarity present in the model embedding space, and so on. The eigen-space projected embedding is then converted to its $(d-1)$-dimensional hyper-spherical angular coordinates, using the mathematical framework described in Section 2.

In terms of implementation, since the first dimension of ε' is the most important, its hyper-spherical angular coordinate value is computed first using Equation 1, and so forth. Thus, the first angular coordinate stores the maximum semantic information from the eigen-transformed ε' embedding. Observe that the ortho-spherical transformation of the original embedding ε is now ordered in terms of semantic information content within the angular values. This enables us to bypass the scenarios wherein angular values pertaining to important embedding dimensions may be truncated.

Based on Observation 1, we note that the lower-end of angular values contribute lesser and lesser to the cosine similarity score between two embedding vectors. Thus, using the obtained ordering of angular coordinates, *embedding dimensionality reduction* can be performed by simply truncating the $(d-1)$ angular values to keep the top-k coordinates, depending on the required compression ratio. The possible retention of partial information from the truncated angular values within the top-k (Observation 2) further ensures, intuitively, limited loss of semantic information.

Computing Cosine Similarity. The retained k angular coordinate values are considered to form the reduced k-dimensional embedding representation from model $\mathcal{M}$ which are stored in a vector database for downstream semantic relevance based applications.

For computing the cosine similarity between two embedding vectors, the corresponding angular values are transformed back to the low-dimensional approximate embeddings using Equation 2, and the vector dot-product is then computed. Observe, that this computation process is extremely efficient as the reconstruction operates only on k angular values, and the vector dot-product is also computed on the low-dimensional reconstructed approximate embeddings.

As an implementation detail, since the original $(d-1)$ angular values are truncated to k, we only use the formulation of x_j (i.e., the left term) in Equation 2 – as the final dimension of the original embedding vector (i.e., x_d) cannot be reconstructed due to possible truncation of θ_{d-1}.

Matryoshka Embeddings. An interesting and extremely beneficial by-product of OySTER is the inherent generation of Matryoshka-like embedding from any model $\mathcal{M}$, without the need for explicit MRL-based training process. Observe that the $d-1$ angular coordinate representation obtained for an embedding vector (after the eigen-space projection), is already sorted in terms of the amount of semantic information contained in them. Thus, we can easily slice any number of top-p angular dimensions (or angular values) depending on the application requirement, and compute the cosine similarity using the reconstructed approximate embedding vectors (from these p coordinate values).

4 Empirical Evaluation & Results

We now present our experimental setup and empirical results to compare the efficacy of OySTER, against existing baselines, with varying embedding dimensionality reduction ratios.

Models and Baselines. We apply OySTER to embeddings obtained from *two* different modern language models, namely *decoder-only* **Qwen-3-Embedding-0.6B** and *encoder-only* **Snowflake-Arctic-embed-l-v2.0** (openly available on Huggingface). Note that both models have been pre-trained with MRL objective (for Matryoshka embeddings). As baselines, we compare the performance of OySTER with the widely-used dimensionality reduction techniques of **PCA** and **UMAP**.

Benchmark and Metrics. We conduct experiments across different datasets and diverse semantic-related tasks of the **MTEB** benchmark Muennighoff et al. (2023). Specifically, we showcase results pertaining to Semantic Relevance, BiTextMining, Summarization, Pair Classification, Reranking, and Retrieval tasks – reporting both the main and secondary performance metric scores.

Table 1: Performance comparison with varying dimensionality of OySTER on **different** semantic tasks of *MTEB* benchmark. [**Note:** *The performance of the original model is marked by a* †.]

Task	Dataset	Evaluation Metric	Method	Qwen3-Embedding-0.6B				Snowflake-Arctic-embed-l-v2.0			
				8	*16*	*32*	*64*	*8*	*16*	*32*	*64*
Reranking	AskUbuntuDupQuestions	*nDCG@10*	*Original*†			67.77				68.73	
			UMAP	57.05	56.38	57.54	57.23	**58.00**	57.03	57.55	58.00
			PCA	56.66	59.11	63.09	65.88	55.12	58.72	62.84	65.77
			OySTER	**57.14**	**60.58**	**63.58**	**66.99**	55.54	**59.46**	**62.98**	**66.11**
		Recall@10	*Original*†			76.35				75.72	
			UMAP	67.04	65.77	67.79	66.86	**68.25**	68.34	66.98	68.23
			PCA	66.43	68.12	71.87	73.29	64.57	67.07	71.12	73.76
			OySTER	**67.71**	**70.56**	**72.85**	**75.60**	66.19	**68.62**	**71.86**	**73.90**
Pair Classification	SprintDuplicateQuestions	*Max. AP*	*Original*†			95.71				96.54	
			UMAP	**68.34**	69.22	65.76	64.10	**75.21**	74.45	71.82	73.02
			PCA	58.30	81.14	92.11	95.37	65.72	83.46	92.54	96.05
			OySTER	60.34	**82.43**	**92.89**	**95.79**	68.28	**86.02**	**93.30**	**96.45**
		Max. F_1	*Original*†			91.55				92.20	
			UMAP	**70.53**	71.04	66.38	66.38	**74.89**	74.04	70.14	70.87
			PCA	55.48	74.79	86.15	91.07	61.55	77.56	86.91	91.86
			OySTER	57.09	**75.43**	**87.35**	**91.55**	62.46	**79.13**	**87.27**	**92.42**
Summarization	SummEval.v2	*Spearman's Correlation*	*Original*†			27.80				33.83	
			UMAP	-0.27	0.37	0.79	-2.21	5.17	3.64	5.45	2.94
			PCA	**17.23**	**21.43**	21.98	23.48	**16.17**	**20.97**	27.67	31.47
			OySTER	16.30	21.08	**22.23**	**24.20**	15.40	19.33	**29.28**	**31.78**
		Pearson's Correlation	*Original*†			33.68				38.81	
			UMAP	-0.50	1.14	3.40	0.39	4.24	3.83	7.60	5.62
			PCA	**22.29**	25.15	27.10	29.27	**22.45**	26.58	32.61	36.07
			OySTER	21.65	**25.28**	**27.41**	**29.33**	22.15	**26.92**	**33.64**	**36.53**
BiTextMining	Tatoeba	*F_1 Score*	*Original*†			48.79				66.85	
			UMAP	**14.29**	15.00	14.98	15.20	**25.87**	26.57	27.09	26.61
			PCA	6.31	20.76	**34.31**	**42.57**	10.70	32.58	50.23	58.89
			OySTER	6.77	**21.54**	34.22	41.91	10.89	**32.97**	**50.48**	**59.59**
		Accuracy	*Original*†			53.10				70.21	
			UMAP	**18.16**	18.92	18.99	19.33	**30.65**	31.25	31.87	31.35
			PCA	9.56	**26.06**	**39.72**	**47.94**	15.05	**38.42**	**55.05**	**63.99**
			OySTER	9.66	**26.06**	38.96	46.59	14.66	38.32	54.95	63.42

Experimental Results. We first present the performance comparison of the different approaches on Reranking, Pair Classification, Summarization, and BiTextMining tasks of MTEB, using a corresponding dataset. As an upper bound, we also report the performance of the original model (i.e., using the full 1024-dimensional embeddings without any modifications from the models). We vary the embedding dimensionality compression ratio from $128\times$ to $16\times$, which corresponds to reducing the original 1024 embedding dimensionality to 8, 16, 32, and 64 – posing a challenging setup. Also, the run time for OySTER was comparable with PCA, due to the efficient reconstruction using Eq. 2.

From Table 1, we see that on most tasks, OySTER performs better than the other baselines, on both performance measures, for a dimensionality reduction ratio of $32 - 64\times$. On noted in literature, UMAP performs well for extremely low dimensions – as in when the embedding dimensionality is reduced to only 8 in our setup. We observe the same trend across the tasks and across the two models (from different families) considered. At higher dimensions (e.g., 64) the performance of dimensionality reduction plateaus, wherein PCA and OySTER depicts comparable performance.

Matryoshka Embeddings. As discussed earlier, an added advantage of OySTER is the inherent generation of Matryoshka-like embeddings (from the ordered hyper-spherical angular values). To this end, we evaluate the performance of our approach against the two baseline models (trained with MRL objective), on the various semantic tasks of MTEB benchmark. Table 2 tabulates the obtained results, showcasing enhanced performance of OySTER across almost all the considered tasks with

Table 2: Performance comparison with varying dimensionality of OySTER against **Matryoshka** embedding models on different semantic tasks of *MTEB* benchmark.

Task	Dataset	Evaluation Metric	Method	Qwen3-Embedding-0.6B			Snowflake-Arctic-embed-l-v2.0		
				8	*32*	*128*	*64*	*128*	*256*
Semantic Rel.	STS-22.v2	*Spearman's Correlation*	*Matryoshka*	51.49	65.43	69.87	65.81	68.87	69.38
			OySTER	**54.75**	**67.65**	**70.54**	**69.17**	**70.39**	**70.33**
		Pearson's Correlation	*Matryoshka*	44.08	60.90	65.98	**62.33**	64.93	66.05
			OySTER	**45.07**	**61.57**	**67.68**	61.80	**64.95**	**66.25**
Retrieval	NFCorpus	*nDCG@10*	*Matryoshka*	3.30	10.08	22.17	12.62	20.95	25.79
			OySTER	**6.79**	**17.51**	**26.67**	**22.57**	**27.69**	**30.58**
		Recall@10	*Matryoshka*	1.60	5.46	10.13	5.29	9.95	12.17
			OySTER	**2.69**	**7.92**	**12.40**	**10.85**	**12.68**	**14.16**
Reranking	AskUbuntuQ.	*nDCG@10*	*Matryoshka*	56.43	**64.83**	67.44	62.81	64.51	64.63
			OySTER	**57.14**	63.56	**67.62**	**64.19**	**65.34**	**65.67**
		Recall@10	*Matryoshka*	66.17	**73.96**	**76.54**	71.36	72.39	72.34
			OySTER	**67.71**	72.85	75.95	**72.00**	**72.87**	**72.57**
Pair Classif.	SprintDupQ.	*Max. AP*	*Matryoshka*	38.97	87.03	94.19	93.96	95.78	96.66
			OySTER	**60.34**	**92.89**	**96.30**	**96.85**	**97.50**	**97.40**
		Max. F_1	*Matryoshka*	42.21	81.79	88.78	88.66	91.50	92.55
			OySTER	**57.14**	**87.35**	**92.32**	**92.91**	**94.27**	**93.73**
BiTextMining	Tatoeba	*F_1 Score*	*Matryoshka*	3.74	26.57	41.22	66.17	69.99	72.23
			OySTER	**6.70**	**34.15**	**46.24**	**69.46**	**72.59**	**73.87**
		Accuracy	*Matryoshka*	5.29	30.66	45.59	69.96	73.29	75.38
			OySTER	**9.57**	**38.88**	**50.73**	**72.78**	**75.73**	**76.93**
Summarizat.	SummEval.v2	*Spearman's Correlation*	*Matryoshka*	13.69	**23.85**	**27.89**	32.85	**34.31**	**34.06**
			OySTER	**16.30**	22.23	25.61	**33.91**	33.21	33.48
		Pearson's Correlation	*Matryoshka*	18.39	**28.19**	**33.39**	36.54	39.00	**39.68**
			OySTER	**21.65**	27.42	31.10	**40.12**	**39.49**	39.12

varying embedding dimension compression. In certain scenarios, like Retrieval and BiTextMining tasks, we obtain nearly $2\times$ improvements over the Matryoshka trained large models.

We further evaluate the approach on a larger set of retrieval tasks, as shown in Table 3. As before, we observe that UMAP performs well at extremely low dimensions (of 8), while OySTER performs better in the other scenarios. With increase in the retained number of dimensions, the performance of PCA and our approach becomes comparable. For example, at 64 dimensions (i.e., $16\times$ reduction), both methods achieve nearly 75% (or higher) of the original model performance.

From Table 4, we observe the empirical evaluation on STS benchmark datasets. For better readability, we report the average results on monolingual and multilingual setting separately. OySTER can be seen to perform better than the other approaches for Qwen3 model, while for the Snowflake model PCA is seen to be slightly better. In both settings, UMAP is seen to perform worse. The difference in the performance trends between the models may be attributed to the disparity in their architectures and training processes (as one is decoder-only and the other is encoder-only).

As an additional use-case scenario, we compare the performances in the high-resource language setting for BiTextMining, as depicted in Table 6. For Tatoeba dataset we consider 5 different languages and the complete BUCC.v2 dataset (as it contains only high-resource languages). Herein, we can observe that OySTER achieves upto 6% F_1 score improvement (in BUCC at $32\times$ compression) over PCA, with an even larger margin over the pre-trained Matryoshka embeddings.

Overall, from the above experiments we observe that our proposed OySTER embedding compression techniques performs better (or comparable) than PCA for $32-64\times$ dimension reduction, along with the added benefit of Matryoshka embedding (on the angular values computed).

Ablation Study. Finally, we present two different ablation study for our methodology. In Table 6(a), we report the results for *quantization*, when the truncated hyperspherical coordinates from OySTER

Table 3: Performance comparison with varying dimensionality of OySTER on **Retrieval** tasks of *MTEB* benchmark. [**Note:** *The performance of the original model is marked by a* †.]

Model	Dataset	Metric / Approach	nDCG@10				Recall@10				MRR@10			
			8	*16*	*32*	*64*	*8*	*16*	*32*	*64*	*8*	*16*	*32*	*64*
Qwen3-Embedding-0.6B (dim=1024)	*ArguAna*	*Original*†			68.11				93.46				60.53	
		UMAP	**31.90**	30.11	30.06	29.83	**55.62**	54.98	53.49	54.48	**24.67**	22.58	22.92	22.34
		PCA	19.25	37.83	52.79	60.14	34.42	60.81	79.59	87.06	14.75	30.96	44.74	52.01
		OySTER	20.03	**40.32**	**55.38**	**62.22**	35.13	**64.01**	**81.15**	**88.12**	15.56	**33.29**	**47.60**	**54.45**
	FiQA	*Original*†			39.03				47.09				46.85	
		UMAP	**7.34**	7.09	7.64	6.92	**11.40**	10.62	11.74	10.73	**9.07**	8.93	9.49	9.07
		PCA	3.02	12.00	19.55	**28.90**	5.22	17.00	25.59	**36.09**	3.59	15.21	**24.88**	**35.55**
		OySTER	3.17	**12.25**	**19.69**	28.26	5.44	**17.49**	**26.62**	35.18	3.73	**15.49**	24.35	34.94
	NFCorpus	*Original*†			29.78				14.77				48.33	
		UMAP	**15.16**	**14.85**	15.43	15.80	**6.30**	5.47	5.90	5.52	**23.79**	22.93	24.10	25.84
		PCA	5.85	10.91	16.28	21.62	2.76	4.35	6.97	9.38	11.61	21.13	29.09	38.02
		OySTER	6.79	13.37	**17.52**	**23.43**	2.69	**5.60**	**7.92**	**10.84**	13.31	**25.84**	**30.51**	**40.34**
	SciDocs	*Original*†			15.75				17.45				26.64	
		UMAP	**7.34**	**7.66**	7.86	7.54	**8.95**	**9.04**	9.07	9.16	**12.55**	**13.23**	14.18	12.82
		PCA	1.98	6.23	10.91	14.16	2.67	7.22	12.42	15.92	3.35	11.42	18.99	24.48
		OySTER	2.22	7.00	**11.81**	**14.76**	2.83	8.20	**13.44**	**16.49**	3.90	12.70	**20.36**	**25.41**
	SciFact	*Original*†			67.84				81.61				63.95	
		UMAP	17.89	19.57	19.61	19.28	29.43	32.87	31.81	30.77	**14.77**	16.41	16.43	16.56
		PCA	17.32	**35.88**	49.41	**58.60**	30.08	**52.24**	63.90	**75.08**	13.62	**31.31**	45.74	**54.17**
		OySTER	**18.03**	35.22	**50.48**	57.86	**30.41**	51.82	**64.18**	73.24	14.54	30.74	**46.68**	53.73
Snowflake-Arctic-embed-l-v2 (dim=1024)	*ArguAna*	*Original*†			57.87				86.91				49.21	
		UMAP	**28.84**	29.20	28.59	29.40	**53.13**	55.41	52.99	53.98	**21.49**	21.30	21.19	21.96
		PCA	16.87	33.77	46.38	52.54	33.00	55.97	73.76	81.08	12.00	26.99	38.11	43.94
		OySTER	17.30	**34.31**	**48.33**	**54.06**	33.29	**57.54**	**75.61**	**82.29**	12.54	**27.20**	**40.08**	**45.55**
	FiQA	*Original*†			38.30				44.44				45.90	
		UMAP	**9.37**	8.03	7.38	7.89	**14.10**	11.58	11.12	11.64	**11.73**	10.32	9.56	10.04
		PCA	2.84	10.16	15.75	25.43	4.70	13.86	19.73	30.76	3.65	13.39	20.88	32.19
		OySTER	3.45	**10.51**	**16.60**	**26.50**	5.30	**14.84**	**21.57**	**32.40**	4.49	**13.75**	**21.62**	**33.81**
	NFCorpus	*Original*†			33.28				15.82				53.13	
		UMAP	**18.05**	**18.83**	**18.43**	18.29	**6.95**	**7.29**	7.38	7.20	**28.59**	**29.64**	29.30	29.44
		PCA	5.35	9.16	14.83	22.21	1.75	3.41	5.46	10.23	11.23	17.57	28.45	40.05
		OySTER	5.92	11.21	17.92	**25.83**	2.50	4.56	**8.22**	**13.03**	12.16	21.88	**32.74**	**43.52**
	SciDocs	*Original*†			18.59				19.95				31.44	
		UMAP	**7.38**	**7.50**	7.67	7.42	**9.05**	**9.48**	9.28	9.16	**12.18**	**11.93**	12.99	12.15
		PCA	2.02	5.79	9.65	13.46	2.45	6.46	10.56	14.34	3.66	10.58	17.46	24.10
		OySTER	2.22	6.08	**10.19**	**14.11**	2.73	6.73	**11.02**	**14.95**	4.07	11.33	**18.64**	**25.06**
	SciFact	*Original*†			69.48				81.69				66.55	
		UMAP	**23.52**	24.36	22.98	24.63	**37.43**	38.41	38.02	38.30	**20.01**	20.59	18.91	20.94
		PCA	12.39	**27.26**	44.79	57.54	20.58	**41.27**	59.66	70.39	10.20	**23.27**	41.18	54.55
		OySTER	13.46	26.77	**47.20**	**59.59**	24.07	39.92	**61.04**	**74.56**	10.52	23.07	**43.95**	**55.84**

are stored in *float16* format (instead of *float32*), and similarly for PCA. We can observe that even with reduced storage precision, the performance of OySTER is, in general, better than PCA.

One important operating principle of OySTER is the initial eigen-projection of the embedding to obtain an ordering of the angular coordinates (Observation 3), that facilitates the truncation operation without severely degrading its performance. To this end, we study the effect of removal of the SVD based projection step in OySTER. From Table 6(b), we can see that this does negatively impact the overall performance across all the considered tasks – validating the intuitions discussed in Section 2.

5 Conclusion

This paper presented OySTER, a novel embedding compression method combining orthogonal eigen-space projection of high dimensional vector embedding (to align with informative dimensions)

Table 4: Performance comparison with varying dimensionality on **Semantic Relevance** tasks of *MTEB* benchmark. [**Note:** *Monolingual STS involves the STS12, STS13, STS15 and STS-B datasets. Multilingual STS involves the STS17 and STS22.v2 datasets. The original model is marked by a †.*]

Dataset	Evaluation Metric	Method	Qwen3-Embedding-0.6B				snowflake-snowflake-arctic-embed-l-v2			
			8	*16*	*32*	*64*	*8*	*16*	*32*	*64*
Monolingual STS	*Spearman's Correlation*	*Original*†			81.96				78.44	
		UMAP	49.99	50.81	49.56	48.99	46.57	48.65	45.67	46.27
		PCA	56.53	67.46	75.11	79.57	**53.01**	**61.75**	**69.63**	**75.09**
		OySTER	**68.92**	**75.00**	**79.06**	**80.65**	52.39	61.32	69.42	74.67
	Pearson's Correlation	*Original*			83.13				77.81	
		UMAP	15.31	15.52	18.58	19.73	9.97	10.89	10.88	10.97
		PCA	52.30	64.15	73.31	79.43	**45.38**	**57.03**	**66.97**	**74.58**
		OySTER	**61.34**	**70.69**	**77.33**	**80.20**	41.94	53.60	63.97	71.52
Multilingual STS	*Spearman's Correlation*	*Original*†			74.52				71.55	
		UMAP	49.12	49.08	49.67	49.72	53.02	52.28	52.99	52.51
		PCA	**52.31**	61.28	68.26	**72.23**	49.64	**57.10**	**65.24**	**69.88**
		OySTER	52.05	**62.28**	**68.74**	71.51	**49.75**	56.54	64.49	69.06
	Pearson's Correlation	*Original*			73.67				69.95	
		UMAP	30.36	30.83	31.18	32.17	30.20	29.39	33.68	34.09
		PCA	45.44	55.48	64.22	**69.63**	**43.49**	**52.45**	**61.15**	**67.14**
		OySTER	**46.58**	**56.64**	**64.71**	68.06	42.22	49.95	58.08	63.98

Table 5: Performance comparison with varying dimensionality of OySTER using Qwen3-Embedding-0.6B model for *BiTextMining* task on *high-resource* languages for: **(a)** *Tatoeba* (with German, Korean, Finnish, Arabic and Vietnamese) and **(b)** BUCC.v2 datasets.

(a)

Tatoeba	Method	F_1 Score			Accuracy		
		8	*32*	*64*	*8*	*32*	*64*
	Matryoshka	7.5	54.9	65.8	10.0	59.9	70.4
	PCA	10.9	63.6	72.5	16.3	69.3	77.1
	OySTER	**14.9**	**67.2**	**74.6**	**19.9**	**71.7**	**78.5**

(b)

BUCC.v2	Method	F_1 Score			Accuracy		
		8	*32*	*64*	*8*	*32*	*64*
	Matryoshka	5.8	76.5	91.3	7.9	81.0	93.3
	PCA	10.8	80.4	95.4	14.2	84.2	96.5
	OySTER	**13.1**	**87.5**	**96.6**	**17.4**	**90.1**	**97.4**

Table 6: Performance of OySTER with varying dimensionality using Qwen3-Embedding-0.6B model on different *MTEB* tasks for: **(a)** *float16* **quantization** level *and* **(b)** Ablation study without SVD (i.e., using on Hyper-spherical transformation on the raw model embeddings).

(a)

Task *(Metric)*	Method	$float16$		
		8	*16*	*32*
NFCorpus *(nDCG@10)*	*PCA*	5.85	10.88	16.26
	OySTER	**6.79**	**13.35**	**17.52**
AskUbuntu *(nDCG@10)*	*PCA*	56.66	59.11	63.09
	OySTER	**57.14**	**60.58**	**63.54**
SprintDupQ *(Max. AP)*	*PCA*	58.30	81.14	92.11
	OySTER	**60.33**	**82.44**	**92.89**
STS-17 *(Spearman's)*	*PCA*	50.20	61.37	**69.38**
	OySTER	**50.30**	**61.73**	68.96

(b)

Task *(Metric)*	Method	Dimensionality		
		8	*16*	*32*
NFCorpus *(nDCG@10)*	OySTER-*SVD*	3.05	3.76	10.19
	OySTER	**5.92**	**11.21**	**17.92**
BUCC.v2 *(F_1 Score)*	OySTER-*SVD*	5.68	29.07	76.13
	OySTER	**13.14**	**47.50**	**87.47**
SprintDupQ *(Max. AP)*	OySTER-*SVD*	42.36	71.84	87.19
	OySTER	**68.28**	**86.02**	**93.30**
SummEval.v2 *(Spearman's)*	OySTER-*SVD*	13.48	15.20	**23.70**
	OySTER	**16.30**	**21.08**	22.23

along with truncated hyper-spherical coordinate transformation. We reported extensive experiments on MTEB benchmark depicting improved performance of our approach in diverse semantic understanding tasks, even under $64\times$ dimensionality reduction ratio. Additionally, we discussed how OySTER inherently provides Matryoshka embeddings for any model without MRL training, and performing better than when trained with Matryoshka learning objective.

AI USE STATEMENT

In this work, we have **not used** generative AI tools in any capacity. The authors confirm that no GenAI technologies were used for content creation, including tables, code, data, and written text.

REFERENCES

Artem Babenko and Victor Lempitsky. Additive Quantization for Extreme Vector Compression. In *CVPR*, 2024.

Neil Burgess, Jelena Milanovic, Nigel Stephens, Konstantinos Monachopoulos, and David Mansell. Bfloat16 Processing for Neural Networks. In *IEEE Symposium on Computer Arithmetic (ARITH)*, pp. 88–91, 2019.

Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(4):744–755, 2014.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating Large-Scale Inference with Anisotropic Vector Quantization, 2020. arXiv:1908.10396.

Moshik Hershcovitch, Andrew Wood, Leshem Choshen, Guy Girmonsky, Roy Leibovitz, Or Ozeri, Ilias Ennmouri, Michal Malka, Sang (Peter) Chin, Swaminathan Sundararaman, and Danny Harnik. ZipNN: Lossless Compression for AI Models. In *IEEE CLOUD*, pp. 186–198, 2025.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

Petr Kasalický, Martin Spišák, Vojtěch Vančura, Daniel Bohuněk, Rodrigo Alves, and Pavel Kordík. The Future is Sparse: Embedding Compression for Scalable Retrieval in Recommender Systems. In *RecSys*, pp. 1099–1103, 2025.

Omar Khattab and Matei Zaharia. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *SIGIR*, pp. 39–48, 2020.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. Matryoshka Representation Learning. In *NeurIPS*, 2022.

Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software*, 3(29):861, 2018.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. MTEB: Massive Text Embedding Benchmark. In *Conference of the European Chapter of the Association for Computational Linguistics*, pp. 2014–2037, 2023.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pp. 583–588, 1997.

Will Trojak and Freddie D. Witherden. Inline Vector Compression for Computational Physics. *Computer Physics Communications*, 258(107562), 2021.

L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Terry Welch. A Technique for High-Performance Data Compression. *Computer*, 17(6):8–19, 1984.

Songhao Wu, Ang Lv, Xiao Feng, Yufei Zhang, Xun Zhang, Guojun Yin, Wei Lin, and Rui Yan. Polarquant: Leveraging Polar Transformation for Key Cache Quantization and Decoding Acceleration. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 50584–50607, 2025.

Han Xiao. Embedding Compression via Spherical Coordinates. In *Geometry-grounded Representation Learning and Generative Modeling (GRaM): ICLR 2026 Workshop*, 2026.

Yuxuan Yue, Zukang Xu, Zhihang Yuan, Dawei Yang, Jianlong Wu, and Liqiang Nie. PCDVQ: Enhancing Vector Quantization for Large Language Models via Polar Coordinate Decoupling, 2025. arXiv:2506.05432.