90% Slimmer Production Embeddings in Just 2 Minutes – Learn How

The Future is Sparse: Embedding Compression for Scalable Retrieval in Recommender Systems

Petr Kasalický [1,2], Martin Spišák [1,3], Vojtěch Vančura [1,2], Daniel Bohuněk [1], Rodrigo Alves [1,2], and Pavel Kordík [1,2]

Recombee [1]

Czech Technical University in Prague, Faculty of Information Technology [2]

Charles University, Faculty of Mathematics and Physics [3]

Introduction

Entity embeddings form the backbone of modern discovery systems. From deep learning recommenders to search engines and vector databases, embeddings encode the fine-grained semantics that power retrieval and ranking at scale. At Recombee, embeddings drive recommendations and search for millions of users across publishers, media platforms, and e-commerce sites worldwide.

The problem is that embedding tables grow along two unforgiving axes: **dimensionality** and **cardinality**. Newer backbones keep producing larger embeddings, while catalogs expand into hundreds of millions or even billions of items. As a rule of thumb, 100M embeddings at 256 dimensions in single precision consume ~100GB of RAM. A routine model upgrade from 512-dimensional SBERT to 768-dimensional Nomic therefore adds another 100GB for every 100M embeddings. Such growth quickly overwhelms inference hardware, making tables hard to fit into memory, slow to scale across clusters, and expensive to serve in real-time pipelines.

Prior attempts to tame this bloat come with trade-offs. **Quantization** shrinks memory but often requires compatible hardware. **Dimensionality reduction** (e.g., PCA, SVD) is simple but risks accuracy loss. **Matryoshka** training enables truncation but necessitates specialized backbone training. And **caching or offloading** systems ease pressure only with significant platform complexity.

We introduce CompresSAE: a sparse autoencoder (SAE) that compresses dense vectors into high-dimensional, sparsely activated representations optimized for retrieval accuracy.

CompresSAE

Architecture

- **Encoder:** non-linear, produces k-sparse embedding. Inputs to the model are normalized

$$\mathbf{s} = f_{enc}(\mathbf{x}; \mathbf{W}_{enc}, \mathbf{b}_{enc}, k) = \phi(\mathbf{W}_{enc}\bar{\mathbf{x}} + \mathbf{b}_{enc}, k)$$

  - $\phi(\cdot, k) : \mathbb{R}^s \to \mathbb{R}^s$ is a function that retains the k entries with the largest absolute values and zeroes out the rest, and serves both as a sparsification mechanism and non-linear activation in our model, replacing the common choice of ReLU and TopK
- **Decoder:** linear, bias-free (enables kernel trick), row-normalized

$$\hat{\mathbf{x}} = f_{dec}(\mathbf{s}; \mathbf{W}_{dec}) = \mathbf{W}_{dec}\mathbf{s}$$

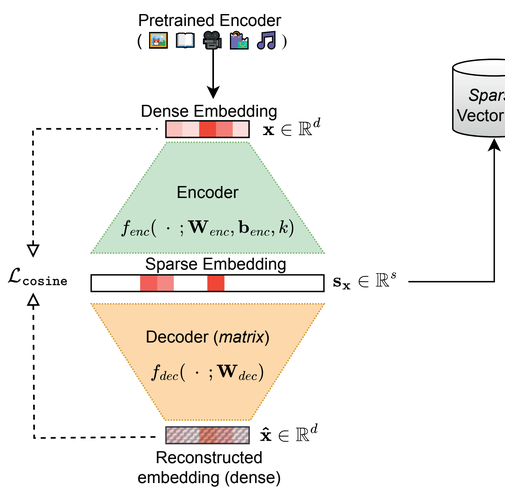
Training

- Train on precomputed embeddings (no raw data/encoder), cosine reconstruction objective

$$\mathcal{L}_{\text{cosine}}(\mathbf{x}, f(\mathbf{x}; \theta, k)) = 1 - \frac{\mathbf{x}^\top \hat{\mathbf{x}}}{\|\mathbf{x}\|_2 \, \|\hat{\mathbf{x}}\|_2}$$

- Hardware: 1x NVIDIA H100 SXM 80GB
- Batch size 100 000, convergence after 500 steps (around 100 seconds)

```python
def phi(t: torch.Tensor, k: int, dim: int = -1) -> torch.Tensor:
        topk = torch.topk(torch.abs(t), k, dim)
        return torch.zeros_like(t).scatter(dim, topk.indices, topk.values) * torch.sign(t)
```



Sparse Embedding Compression

- Encoder maps a dense d-dimensional embedding to a sparse s-dimensional vector with k non-zero entries.
- A batch of b embeddings is mapped to a b x s sparse matrix in CSR format
  - k non-zero entries per row, 2*k*4 bytes per embedding
  - Compressing a 768-dimensional dense embedding into a 4096-dimensional sparse vector with 32 nonzeros yields an effective 12× compression

Retrieval

Choose Your Retrieval Strategy

- **Speed:** Directly from sparse space
- **Accuracy:** From reconstructed space using the kernel trick

Retrieval from Sparse Compressed Space

Sparse compressed embeddings unlock major efficiency gains:

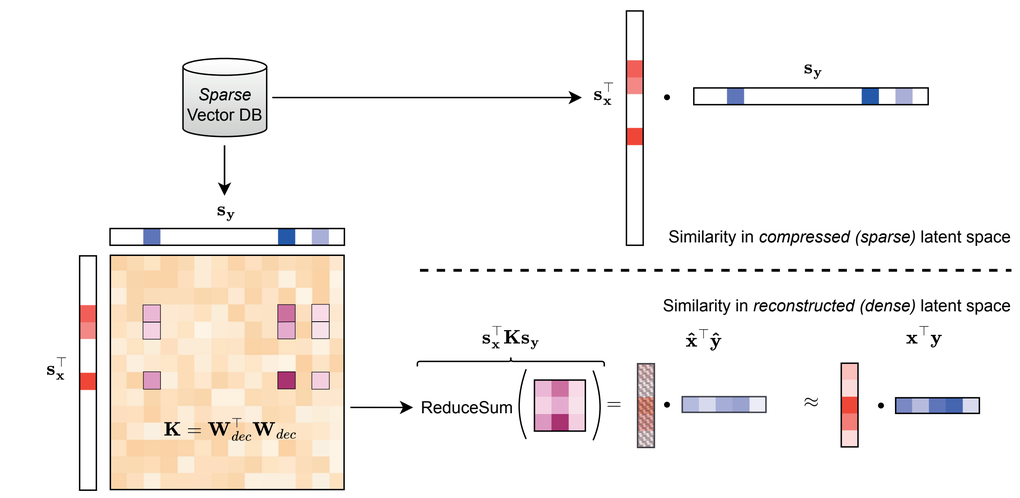
- Dot products between vectors with k nonzero entries have O(k) complexity.
- Cosine similarity computed using efficient sparse matrix–vector (SpMV) kernels.

Retrieval from Reconstructed Space

- **Kernel trick:** retrieve from reconstructed space

$$\frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|_2 \, \|\mathbf{y}\|_2} \approx \frac{\hat{\mathbf{x}}^\top \hat{\mathbf{y}}}{\|\hat{\mathbf{x}}\|_2 \, \|\hat{\mathbf{y}}\|_2} = \frac{(\mathbf{W}_{dec}\mathbf{s_x})^\top (\mathbf{W}_{dec}\mathbf{s_y})}{\|\mathbf{W}_{dec}\mathbf{s_x}\|_2 \, \|\mathbf{W}_{dec}\mathbf{s_y}\|_2} = \frac{\mathbf{s_x}^\top \mathbf{K} \mathbf{s_y}}{\sqrt{\mathbf{s_x}^\top \mathbf{K} \mathbf{s_x}} \sqrt{\mathbf{s_y}^\top \mathbf{K} \mathbf{s_y}}}$$

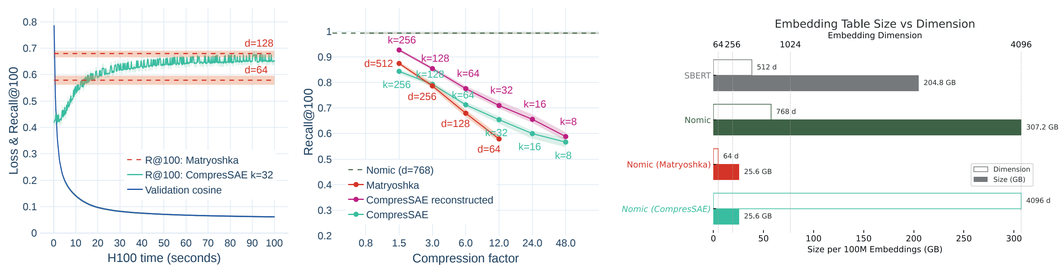
- More accurate (directly optimized via the objective function), more expensive at O(k^2)



Results

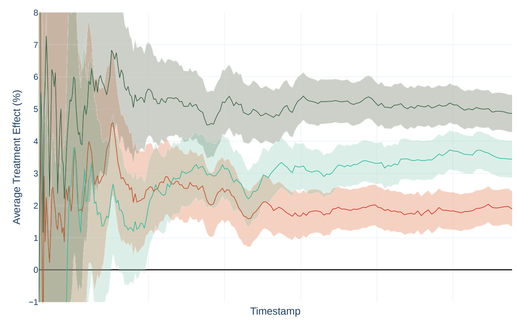
Offline

- **Fast convergence:** CompresSAE surpasses Matryoshka within ~15s.
- **Accuracy vs. compression:**
  - Better balance, especially at high compression.
  - Matches Matryoshka models up to **4× larger**.
- **Reconstructed space retrieval:** best overall trade-off.



Online

- **A/B test:** ~8.5M users per variant.
- **Models compared:**
  - SBERT (512 dim, baseline)
  - Nomic (768 dim, Matryoshka loss)
  - Matryoshka (64 dim)
  - CompresSAE (4096 dim sparse, k=32 nonzero) using sparse retrieval



- **Findings:**
  - Both compressed variants beat the **8× larger SBERT**.
  - CompresSAE achieves 12× compression with only a small **1.35% CTR** drop.
  - CompresSAE outperforms equal-size Matryoshka by +**1.52% CTR (stat. sig.)**.

Paper
(ACM Digital Library)

Reference Implementation
(GitHub)

Acknowledgements:
This research was supported by the Czech Science Foundation (GAČR) project 25-16785S and the Grant Agency of the Czech Technical University in Prague under grant SGS23/210/OHK3/3T/18.