Recombee

Sparse Representation Learning

Compression, interpretation, and recommendation

Vojtech Vancura · 2.11.2026



A large vocabulary, a small description



An item can have many possible features.
Only a few need to describe this particular item.

What we want	What sparsity can give us
Smaller embeddings	Store active coordinates and their values
Understandable representations	Inspect items sharing a feature
Better recommendations	Learn which distinctions matter in a domain
Smaller models	Learn sparse embedding parameters directly

The same idea appears in **compression, control, retrieval, and model training**.

What is a sparse vector?



A vector $z \in \mathbb{R}^h$ is **k-sparse** when at most k entries are nonzero.

Coordinate	0	1	2	3	4	5	6	7
Dense	–.90	.12	.70	–.04	.55	.01	–.60	.08
Sparse	**–.90**	0	**.70**	0	0	0	**–.60**	0

$$\|z\|_0 = |\{j : z_j \neq 0\}| = 3$$

Sparse storage: `indices = [0, 2, 6]`, `values = [-.9, .7, -.6]`.

Illustrative example. A sparse representation can contain negative values.

Width and active budget are different choices



Width h

How many features are available?

A larger dictionary can distinguish more patterns.

Budget k

How many features can one item use?

A smaller budget limits storage and forces selection.

$$h = 4096, \quad k = 32 \quad \Rightarrow \quad 0.78\% \text{ active coordinates}$$

Increasing width need not increase the number of stored values per item.

How can we learn sparse representations?



Approach	What we optimize or constrain
Sparsity penalty	Reconstruction loss $+\lambda \|\|z\|\|_1$
Thresholding	Keep activations above a threshold
Per-vector top-k	Keep a fixed number of selected coordinates
Parameter pruning	Remove weights during model training

A penalty controls sparsity indirectly. A per-vector top-k gate gives a predictable active budget.

Today: **absolute top-k** for signed sparse codes, then sparse parameters.

Sources: PuRL, §2–3; Compressed ELSA, §2; Compresso, §2.

A sparse autoencoder learns a dictionary



$$x \in \mathbb{R}^d \xrightarrow{\text{encoder}} a \in \mathbb{R}^h \xrightarrow{\text{sparsify}} z \in \mathbb{R}^h \xrightarrow{\text{decoder}} \hat{x} \in \mathbb{R}^d$$

The encoder chooses **which features** describe x and **how strongly**.

With a linear decoder:

$$\hat{x} = W_{\text{dec}} z = \sum_{j \in \text{supp}(z)} z_j w_j$$

Each decoder column w_j is a learned direction. Reconstruction trains the dictionary and the encoder together.

Sources: The Future is Sparse, §2; SAE Adapters, §2. Bias terms omitted for clarity.

Absolute top-k keeps magnitude and sign

$$J = \mathrm{TopKIndices}(|a|, k), \qquad z_j = \begin{cases} a_j & j \in J \\ 0 & j \notin J \end{cases}$$

Activation	−.90	.12	.70	−.04	.55	.01	−.60	.08
AbsTopK, $k = 3$	**−.90**	0	**.70**	0	0	0	**−.60**	0

Rank by absolute value; keep the original signed value.

ReLU + top-k would discard the negative directions. AbsTopK can use both poles of a feature.

Selects k slots; actual nonzeros can be fewer if selected values are zero. Ties need a consistent implementation rule.

Source: The Future is Sparse, §2. PuRL instead evaluates nonnegative ReLU + top-k codes.

Training needs more than a hard gate



Match the loss to the representation's use. For directional retrieval:

$$\mathcal{L}_{\cos} = 1 - \frac{x^\top \hat{x}}{\|x\|_2 \|\hat{x}\|_2}$$

- Hard selection changes discontinuously; gradients through retained values do not differentiate the choice of indices.
- Compresso provides **straight-through** top-k operators: surrogate gradients for a hard forward pass.
- Features that rarely win can stop learning. Monitor feature usage, reconstruction, and the downstream task.

Sources: The Future is Sparse, §2 (including a 4k auxiliary reconstruction); Compresso, §2. A gradient estimator is not an exact derivative of selection.

Compression means counting bytes

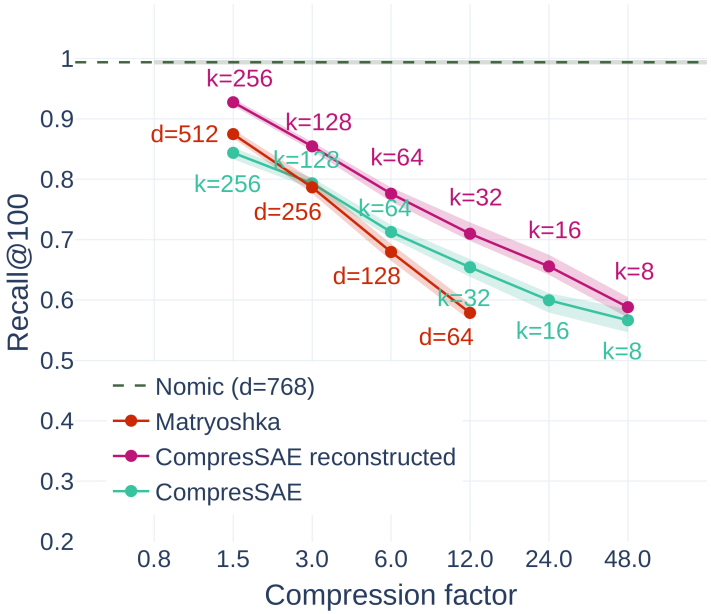
One item

Dense: **768 float32 values**.

Sparse: **32 values + 32 indices**, in a 4096-dimensional space.

$$\frac{768 \times 4}{32 \times (4+4)} = 12\times$$

3072 → 256 bytes per item.



At **100M items**: **307.2 GB → 25.6 GB** of vector payload.

Float32 values, int32 indices; decimal GB. Excludes row pointers, model weights, indexes, and runtime buffers.

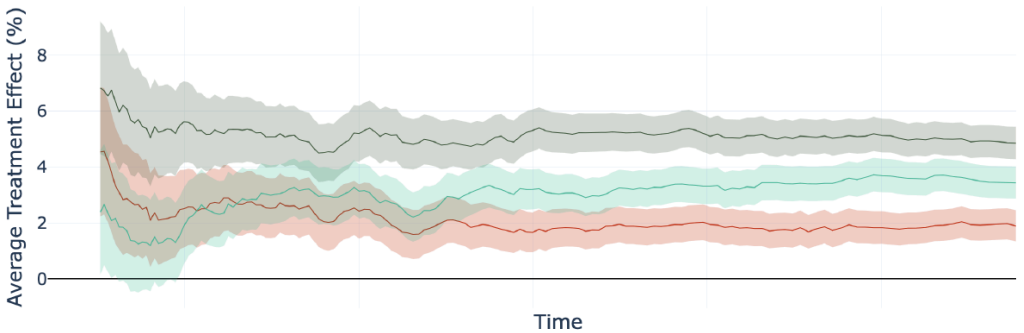
The Future is Sparse (RecSys 2025), §2.2 and Figure 3, center panel. Original paper figure.

The Future is Sparse: a production trade-off

Candidate embeddings changed; **downstream recommendation CTR** was measured online.

Model (Compression)	Embedding Dimension	CTR Lift	Size per 100M Embeddings
SBERT [18]	512	(baseline)	204.8 GB
Nomic [14]	768	+4.86%	307.2 GB
Nomic (Matryoshka)	64	+1.89%	25.6 GB
Nomic (CompreSSAE)	4096*	**+3.44%**	25.6 GB

*Sparse embeddings with 32 nonzero entries.

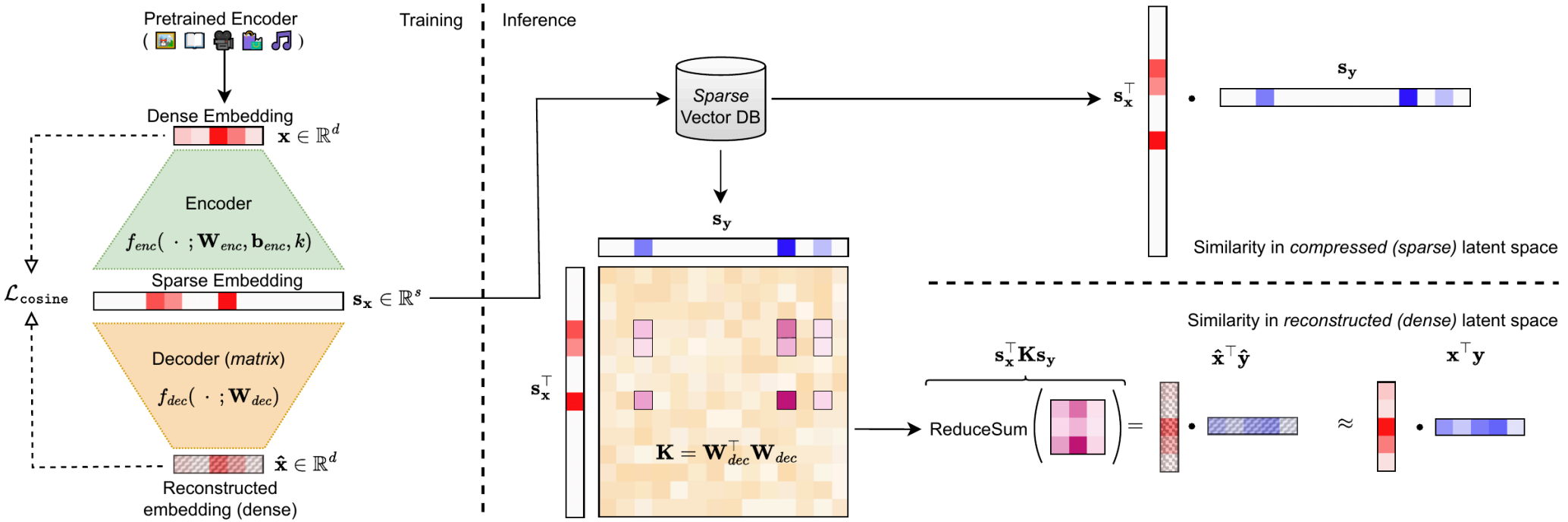


Same 25.6 GB payload: sparse compression retains more of Nomic's benefit than the 64-dimensional Matryoshka representation.

One proprietary media setting; about 8.5M users per variant. Curves show anytime-valid 99% confidence intervals. Relative CTR lifts over SBERT, not percentage-point changes.

Kasalický et al., The Future is Sparse (RecSys 2025), Figure 1 and §3. Paper

Two retrieval geometries



Sparse: $z_x^\top z_y$. **Reconstructed:** $z_x^\top K z_y$, with $K = W_{\text{dec}}^\top W_{\text{dec}}$.

Different geometries. Reconstructed cosine also needs normalization; the kernel adds storage and up to k² cross-terms per pair.

The Future is Sparse (RecSys 2025), Figure 2 and §2.2. Original paper figure.

From a coordinate to a concept



An active coordinate gives us something concrete to inspect:

1. Find items with strong activation on that feature.
2. Inspect examples and compare against metadata.
3. Assign a tentative label, keeping positive and negative poles separate for signed codes.
4. Change the feature and check what happens downstream.

A readable label is a hypothesis about a feature. An intervention tests whether it is a useful control.

Sources: PuRL, §3–4; Telegraph, §2; SAE Adapters, qualitative analysis. No one-concept-per-neuron guarantee is assumed.

PuRL: discover levers in collaborative models



$$\text{history} \xrightarrow{E_c} u \xrightarrow{E_s} z \xrightarrow{D_s} \hat{u} \xrightarrow{D_c} \text{recommendations}$$

- Train the collaborative model from **user–item interactions**.
- Fit an SAE to its user representations; keep the collaborative model fixed.
- Associate neurons with concepts using item tags **after representation learning**.
- Expose the mapped features as recommendation controls.

Concepts are discovered in a collaborative space, rather than supplied as training labels.

Spišák et al., Pulling the Right Levers (PuRL), §3. ELSA and MultVAE; nonnegative top-k SAE.

Steering is a blend, not an item filter



Let $\bar{z}_u$ be the user's sparse code normalized to unit sum. Let v_c encode a desired concept.

$$z'_u = (1-\alpha)\bar{z}_u + \alpha v_c, \qquad 0 \le \alpha \le 1$$

Steering strength	Interpretation
$\alpha = 0$	Original reconstructed profile
Small α	Personal taste, with more emphasis on the concept
$\alpha = 1$	Concept vector alone

A concept can map to **several neurons**. PuRL finds that using several can improve the steering trade-off.

Source: PuRL, §3.3 and §4.4. This blend uses PuRL's nonnegative codes; it is not a blanket rule for signed AbsTopK codes.

PuRL: the same user, different emphasis



The example uses $\alpha = .15$ and the same underlying user profile.

- “Love story” adds romantic films consistent with the user's taste for classics and musicals.
- “Children” and “David Lynch” emphasize different parts of the catalogue.

Control changes the recommendation inside the model. This is a qualitative example, not a global accuracy result.

Source: PuRL, Figure 4 and §4.4. Original figure from the supplied paper.

Control depends on reconstruction fidelity



97%

of original **Recall@20 and nDCG@20** retained in one PuRL setting.

ELSA, MovieLens-25M, 1024-dimensional input, $k = 32$, cosine reconstruction.

The failure case matters

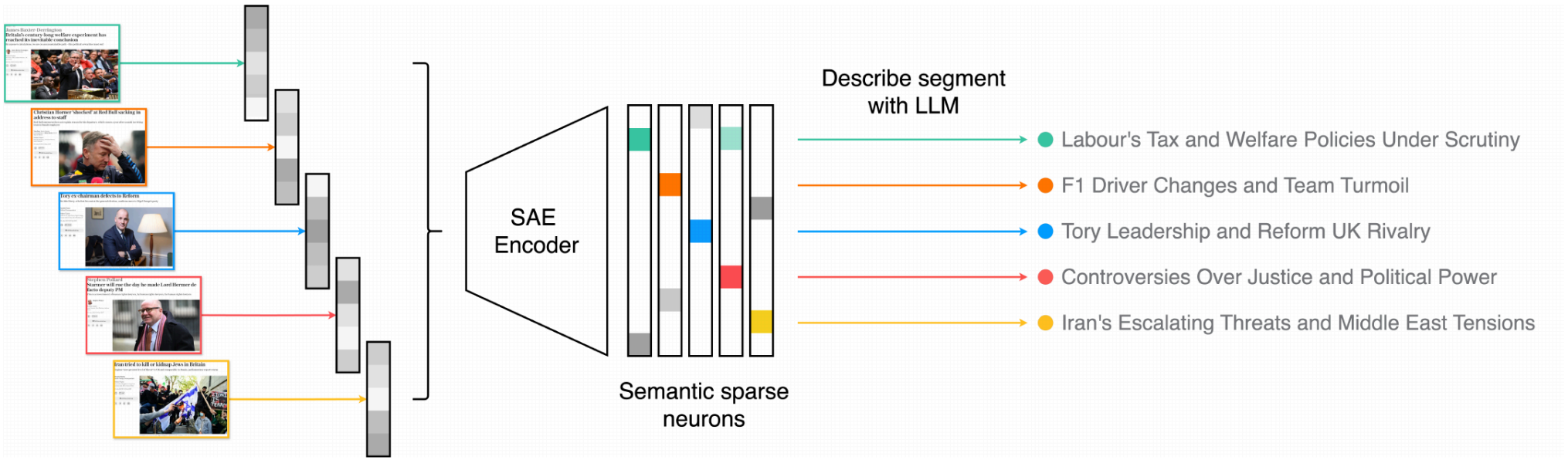
Cosine reconstruction works well for ELSA's directional embeddings.

The same loss substantially degrades MultVAE's downstream quality.

Validate recommendation fidelity before turning sparse coordinates into user controls.

Source: PuRL, §4.2. 97% is performance retained, not a 97% absolute recommendation metric.

The Telegraph: features become news segments

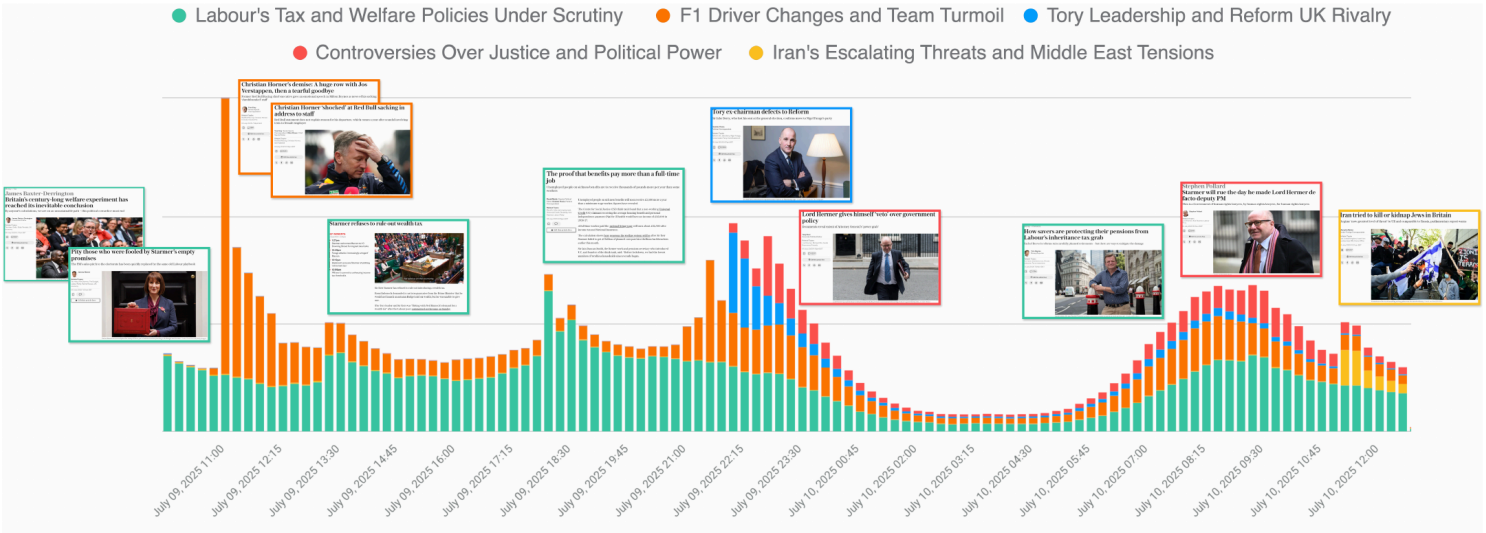


Shared activations define **overlapping article groups**. An LLM names the groups using headlines and summaries.

One article can belong to several stories. The segments become queryable objects in the recommendation platform.

Spišák et al., Segment-Aware Analytics… Lessons from The Telegraph, §2 and Figure 2 (INRA 2025). Original figure.

The Telegraph: make the features timely



- Sample training articles by **interaction volume and recency**, rather than uniformly.
- Refresh segmentation **every 15 minutes** in the reported deployment.
- Let editors inspect segment trends, engagement, and conversions.

Source: Telegraph, §2–3 and Figure 3. Deployed analytics workflow; no controlled CTR uplift from segmentation is established.

An SAE can also be a domain adapter



A general embedding model knows about many domains.
Your recommender needs distinctions within one of them.

$$\text{item text} \xrightarrow{\text{frozen embedder}} s_i \xrightarrow{\text{SAE on this catalogue}} z_i$$

No embedder finetuning. No interaction labels in the adapter's training loss.

The SAE reconstructs item embeddings under a sparse constraint. Its codes are used directly for content-based recommendation.

Vančura et al., Sparse Autoencoders as Semantic Domain Adapters for Recommender Systems (RecSys 2026), §2.

Test adaptation on unseen items



13 datasets × 3 embedders × 2 metrics = 78 evaluation settings.

Step	Protocol
Split items	85% train / 5% validation / 10% test
Train adapter	Reconstruct training-item embeddings only
Select hyperparameters	Validation retrieval performance
Build user profiles	Interactions with training items
Evaluate	Held-out test items, Recall@20 and NDCG@20

$$\hat{x}_u = x_u Z Z^\top$$

Source: SAE Adapters, §2–3. BGE, Qwen3-Embedding-0.6B, MiniLM-L6; 13 datasets. Interactions are used for selection and evaluation.

Sparse adapters win most tested settings

70 / 78

settings have an SAE variant as the **best tested adapter**.

Best variant	Settings
Denoising SAE	**46**
SAE	**24**
β-VAE	8

Dense autoencoders also improve over raw embeddings. **Domain adaptation is part of the gain; sparsity is an additional design choice.**

Signed AbsTopK codes, row-wise L1 normalization. Preliminary study; wins across settings are not independent statistical replications.

Source: SAE Adapters, §2–3 and Figure 1. Counts of best results, not percentage improvements.

TopK: sparsity makes richer retrieval practical



Multi-vector retrieval compares **sets of token embeddings**:

$$\mathrm{MaxSim}(Q, D) = \sum_i \max_j q_i^\top d_j$$

Sparse Multi-Vector Encoding (SMVE):

1. Project tokens onto random anchor directions.
2. Keep the largest k values per token; pool into a sparse descriptor.
3. Retrieve candidates sparsely; re-rank with the original multi-vector score.

Not a trained SAE; uses raw top-k, not absolute top-k. Sparsity makes an expressive score affordable as a retrieval pipeline.

TopK, SMVE: Multi-Vector Retrieval That Just Works, March 2026. Method overview, not a cross-paper benchmark.

Sparse activations and sparse parameters

	Sparse activations	Sparse parameters
What is sparse?	The code for an input	The learned model weights
What changes with input?	Active coordinates and values	Weights stay fixed at inference
Example	$z = \mathrm{AbsTopK}(Wx + b)$	An item table with k entries/row
What gets smaller?	Stored codes; downstream work	Model storage; supported operations

A sparse code does not make the encoder and decoder sparse.

An item table is both a collection of vectors and model parameters. Compressed ELSA learns that table sparsely.

Sources: Compresso, §2; Compressed ELSA, §2. A user vector pooled from sparse item rows may have more than k nonzeros.

Compressed ELSA learns sparse item rows



ELSA learns an item table A. A user profile pools the rows of the items in their history:

$$u = x^\top A, \qquad \hat{r}^\top = uA^\top - x^\top$$

Substitute a row-normalized sparse table:

$$A_s = \mathrm{rowL2}(\mathrm{AbsTopK}(A, k))$$

$$\min_A \mathcal{L}\big(X,\ X(A_s A_s^\top - I)\big)$$

Train for recommendation quality with sparsity in the forward pass, rather than compressing a finished model.

Vančura et al., Efficient Learning of Sparse Representations from Interactions (WWW 2026), §2. A is dense during training before final sparse export.

Start wide, then prune gradually

$$k_0 = h \quad \longrightarrow \quad k_1 \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad k_T = k$$

Early: let dimensions participate while the model discovers useful structure.

Later: reduce active entries per row and adapt to the smaller budget.

The paper compares constant, step-wise, linear, and exponential schedules, with and without restarting surviving weights from their original initialization.

Finding the final sparse model can still require dense training.

Source: Compressed ELSA, §2–3 and Figure 1. Frequent gradual pruning beats waiting for full convergence in the reported experiments.

The lottery ticket hypothesis



A randomly initialized dense network contains a **sparse subnetwork** that can reach comparable test accuracy when trained alone, in a similar number of iterations.

Finding a “winning ticket”	What happens
Train	Learn weights in the dense network
Prune	Remove small-magnitude weights
Reset and retrain	Restore surviving weights to their **original initialization**

The ticket is the sparse connections and their initial weights.

This motivates testing weight resets in Compressed ELSA; it does not guarantee that pruning any model preserves quality.

Frankle & Carbin, ICLR 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks.

Compressed ELSA: compare at the same size



Goodbooks-10k, nDCG@100. Float32 values and int32 indices.

Model	1024 B/item	256 B/item	64 B/item
Low-dimensional ELSA	.433	.357	.270
ELSA + post-hoc SAE	.436	.420	.408
Pruned EASE	.478	.473	.458
Compressed ELSA	**.491**	**.483**	**.469**

Dense ELSA: **.489 at 13,000 B/item**. At 256 B/item, about **51× smaller**, with **1.2% lower nDCG**.

Selected dataset and three budgets from Table 1; no universal compression ratio. The paper also evaluates MovieLens-20M and Netflix Prize.

Source: Compressed ELSA, Table 1. 13,000/256; (.489–.483)/.489. Excludes runtime overhead.

The same latent space can rank segments



Group items by **dominant signed feature**, then describe and merge compatible groups.

Construct segment vectors B_s in the space of item vectors A_s:

$$\underbrace{x^\top A_s}_{\text{one user profile}} \qquad \begin{cases} A_s^\top & \text{rank items} \\ B_s^\top & \text{rank segments} \end{cases}$$

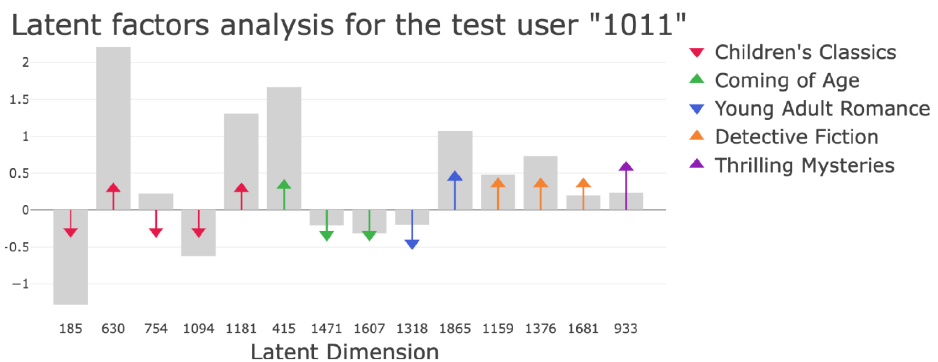
The demo connects **history** → **latent factors** → **recommended segments** → **items**.

Factors are learned from interactions. Segment names use metadata and an external semantic model. Interpretability evidence is qualitative, centered on Goodbooks-10k.

Source: Compressed ELSA, §2–3 and Figure 2.

Demo: Compressed ELSA

Open the Compressed ELSA demo



Paper example: user 1011. Gray bars are user activations; arrows mark segment factors.

1. Inspect a user's history and activations.
2. Compare recommended segments.
3. Inspect items; try a second user.
4. Explore size and pruning results.

Watch for: compact weights, readable segments, and recommendations connected by one latent space.

Compressed ELSA, Figure 2. Repository.

Sparse tensors in PyTorch



Representation	Useful property
Dense tensor containing zeros	Ordinary operators; still stores the zeros
COO	Explicit coordinates and values
CSR / CSC	Compressed row / column structure
Fixed-k row-packed	Predictable indices and values per row

Storage, operators, and autograd support must line up. A conversion alone does not make training efficient.

Measure actual shapes, device, dtype, and sparsity. Include conversion and indexing costs.

PyTorch sparse docs; Compresso, §2. Row-packed is Compresso's SRPTensor, not a built-in layout.

Compresso: fit once, encode new items



```python
from compresso import TopKSAEConfig, TopKSAETrainer

trainer = TopKSAETrainer(
    TopKSAEConfig(hidden_dim=4096, k=32)
)
trainer.fit(train_embeddings)
codes = trainer.transform(new_embeddings)
```

`codes` is an **SRPTensor**: feature indices, values, and a logical dense shape.

New items use the **same learned dictionary**.

Documented API; inputs are compatible dense matrices. Selection by raw score or absolute magnitude is configurable.

Compresso, §2; README API. Install: `pip install compresso-pytorch`.

Compresso building blocks

Need	Building block
Learn and encode sparse codes	`TopKSAETrainer`, `TopKSAEConfig`
Store fixed-k rows	`SRPTensor`
Hard top-k with surrogate gradients	`topk_ste`, `TopKSparsify`
Train values on fixed sparse support	`SRPParam`
Discover sparse weights by pruning	`MaskedParam`, `SparsityController`
Inspect recurring factors and groups	`ClusteringPipeline`

Start with the trainer. Use pruning and custom operators when the model itself needs to be sparse.

Vančura et al., COMPRESSO: Espresso-Style Sparse Representation Learning for Interpretable Recommender Systems (RecSys 2026), §2.

Choose what sparsity should buy you



Goal	What to validate
Compression	Total bytes and retrieval quality
Explanation and control	Feature coherence, fidelity, intervention effects
Better recommendations	Held-out metrics against adapted dense baselines
Smaller model	Parameter storage and measured inference cost

A large feature space with a small active budget is a useful design freedom.

The objective and evaluation decide whether it helps.

Papers behind the story



- **Compression:** Kasalický et al. (2025). *The Future is Sparse: Embedding Compression for Scalable Retrieval in Recommender Systems.* RecSys / arXiv.
- **Control:** Spišák et al. *Pulling the Right Levers: Steerable Collaborative Autoencoders with Sparse Autoencoders.* Paper. Code.
- **Editorial analytics:** Spišák et al. (2025). *Segment-Aware Analytics for Real-Time Editorial Support in Media Groups: Lessons from The Telegraph.* INRA paper.
- **Domain adaptation:** Vančura et al. (2026). *Sparse Autoencoders as Semantic Domain Adapters for Recommender Systems.* RecSys.

Models, retrieval, and tools



- **Sparse parameters:** Vančura et al. (2026). *Efficient Learning of Sparse Representations from Interactions.* WWW. Compressed ELSA.
- **Framework:** Vančura et al. (2026). *COMPRESSO: Espresso-Style Sparse Representation Learning for Interpretable Recommender Systems.* RecSys. Code.
- **Multi-vector retrieval:** Spišák and Galovic, TopK (March 2026). *SMVE: Multi-Vector Retrieval That Just Works*. Author technical article.
- **Tensor formats and operators:** Official PyTorch sparse docs.