DB-KSVD: SCALABLE ALTERNATING OPTIMIZATION FOR DISENTANGLING HIGH-DIMENSIONAL EMBEDDING SPACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Dictionary learning has recently emerged as a promising approach for mechanistic interpretability of large transformer models. Disentangling high-dimensional transformer embeddings, however, requires algorithms that scale to high-dimensional data with large sample sizes. Recent work has explored sparse autoencoders (SAEs) for this problem. However, SAEs use a simple linear encoder to solve the sparse encoding subproblem, which is known to be NP-hard. It is therefore interesting to understand whether this structure is sufficient to find good solutions to the dictionary learning problem or if a more sophisticated algorithm could find better solutions. In this work, we propose Double-Batch KSVD (DB-KSVD), a scalable dictionary learning algorithm that adapts the classic KSVD algorithm. DB-KSVD is informed by the rich theoretical foundations of KSVD but scales to datasets with millions of samples and thousands of dimensions. We demonstrate the efficacy of DB-KSVD by disentangling embeddings of the Gemma-2-2B and Pythia-160M models and evaluating on six metrics from the SAEBench benchmark, where we achieve competitive results when compared to established approaches based on SAEs. By matching SAE performance with an entirely different optimization approach, our results suggest that (i) SAEs do find strong solutions to the dictionary learning problem and (ii) that traditional optimization approaches can be scaled to the required problem sizes, offering a promising avenue for further research. We provide an implementation of DB-KSVD at `https://github.com/RomeoV/KSVD.jl`.

1 INTRODUCTION

Large transformer models have enabled a wide range of applications in natural language processing (Vaswani et al., 2017), computer vision (Dosovitskiy et al., 2021), and numerous other fields (Dalla-Torre et al., 2025; Wen et al., 2023; Brohan et al., 2023). To deploy these models in high-stakes applications, it is desirable to understand their internal representations to validate their reasoning and detect potential biases in their world models. However, interpreting these internal representations remains a significant challenge because they cannot easily be decomposed into monosemantic features. Instead, it has been hypothesized that these internal representations live in a highly-entangled vector space, in which monosemantic features are superimposed on top of each other (Elhage et al., 2022).

In this work, we focus on the disentanglement of these internal representations with the goal of mapping entangled latent embeddings to their corresponding monosemantic features. More specifically, we aim to learn one dictionary D such that any given latent embedding y can be expressed as a linear combination of the dictionary elements (columns of D), i.e., $y = Dx$, where x is a sparse vector. In this formulation, the dictionary D serves as an overcomplete basis for the embedding space, and the sparse vector x encodes the monosemantic features of the embedding.

Given a set of samples $\{y_i\}_{i\in 1..n}$, this process of learning the dictionary D and the sparse vectors $\{x_i\}_{i\in 1..n}$ is referred to as dictionary learning and has various applications such as signal recovery and image compression (Tošić & Frossard, 2011). Although this problem is known to be NP-hard (Razaviyayn et al., 2014), a class of dictionary learning algorithms based on alternating optimization (AO) has been proposed with convergence guarantees under some assumptions (Agarwal et al., 2016). However, these guarantees require strong assumptions that typically do not hold in practice.

Nevertheless, proposed algorithms for approximate solutions (Aharon et al., 2006; Rubinstein et al., 2008; Irofti, 2014; Lillelund et al., 2022; Spielman et al., 2012; Patel & Chellappa, 2011; Zhang et al., 2015) have been widely studied in the context of small to medium scale problems, but they do not immediately scale to the large, high-dimensional datasets that we encounter in the context of transformer models.

To apply dictionary learning to the disentanglement of transformer embeddings, we therefore need to scale dictionary learning algorithms to large problem sizes. Recent work has explored the use of sparse autoencoders (SAEs) as a scalable alternative to established dictionary learning algorithms (Bricken et al., 2023; Templeton et al., 2024; Cunningham et al., 2023; Gao et al., 2024; Bussmann et al., 2025). SAEs typically comprise a simple linear encoder and decoder layer and are trained using gradient descent. However, it is not clear whether the solutions found by SAEs approach the global optimum or whether previously proposed AO approaches may perform better. For this reason, we introduce DB-KSVD, an adaptation of the classic AO-based KSVD algorithm (Aharon et al., 2006) that can be applied to datasets consisting of millions of samples with thousands of features.

Our contributions are as follows: **(i)** We present DB-KSVD, a **scalable adaptation of the KSVD algorithm** that can be applied to large-scale datasets and scales well with CPU and GPU availability. **(ii)** We examine **theoretical aspects of the tractability of the dictionary learning problem** in terms of sampling complexity, sparsity, and the number of identifiable features and link this analysis to design considerations for our algorithm. **(iii)** We adopt a key idea from the recent work on SAEs, the **Matryoshka encoding structure** Bussmann et al. (2025), to DB-KSVD and show performance improvements in some metrics. **(iv)** We apply DB-KSVD to disentangle embeddings of the Gemma-2-2B model Mesnard et al. (2024) and **show competitive performance of our learned dictionaries when compared to established SAE-based approaches** on six metrics of the SAEBench benchmark.

2 Preliminaries

The goal of dictionary learning, also known as sparse coding, is to model a set of data samples as linear combinations of a few elementary signals. Formally, given a data matrix $Y = [y_1, \dots, y_n] \in \mathbb{R}^{d \times n}$, we want to find a dictionary $D = [d_1, \dots, d_m] \in \mathbb{R}^{d \times m}$ where $\|d_i\|_2 = 1$ and a sparse coefficient matrix $X = [x_1, \dots, x_n] \in \mathbb{R}^{m \times n}$ where $\|x_i\|_0 \leq k$ such that

$$Y \approx DX \tag{1}$$

and $n \gg m > d$. Specifically, we consider the optimization problem

$$\min_{D,X} \|Y - DX\|_F^2 \quad \text{s.t. } \|d_i\|_2 = 1, \;\; \|x_i\|_0 \leq k. \tag{2}$$

In the context of disentangling transformer embeddings, the data matrix Y consists of the embeddings of the transformer model, and we are trying to find an overcomplete dictionary D with columns that represent monosemantic features of the embeddings.

One family of algorithms solves the optimization problem in Eq. (2) using AO of two subproblems: (i) finding X for a fixed D and (ii) improving D for a fixed structure of X. We refer to these two steps as the sparse encoding step and the dictionary update step, respectively. In this work, we adapt the KSVD algorithm (Aharon et al., 2006), which uses these AO steps. The remainder of this section provides a brief overview of the KSVD algorithm and its two main components.

Sparse Encoding. The sparse encoding step aims to find a sparse vector x_i for each data sample y_i given a fixed dictionary D. This problem is NP-hard (Natarajan, 1995) due to the combinatorial nature of the sparsity constraint, and we must rely on approximate solutions. A number of algorithms have been proposed, including Matching Pursuit (MP) (Mallat & Zhang, 1993), Orthogonal Matching Pursuit (OMP) (Pati et al., 1993), LASSO regression (Tibshirani, 1996), and solving a mixed-integer program (Liu et al., 2019). In this work, we adapt the MP algorithm, which greedily selects dictionary elements d_j and repeatedly updates a residual r as

$$r \leftarrow r - \langle r, d_j \rangle d_j \quad \text{where } j = \arg\max_{j'} \langle r, d_{j'} \rangle \tag{3}$$

until the sparsity constraint becomes active. Using the computational modifications introduced in Section 4.1, we can typically solve the sparse coding problem in less than one millisecond per sample, even for large dictionaries.

Dictionary Update. The dictionary update step updates each dictionary element d_j by considering a subset of the error matrix that contains the residuals of the samples that use the jth dictionary element. Formally, we update each column d_j using the error matrix

$$E_{\Omega_j} = Y_{\Omega_j} - DX_{\Omega_j} \tag{4}$$

where Ω_j denotes the set of column indices of the samples that use the jth dictionary element. We then replace d_j with the first left singular vector of the error matrix $E_{\Omega_j} - d_j X_{j,\Omega}$, i.e., the error matrix without the contribution of the jth dictionary element. For the KSVD algorithm, we also update the values of the nonzero elements in the corresponding row $X_{j,\Omega}$ using the first right singular vector and singular value. Using this process, all dictionary elements are updated sequentially. Although the dictionary update step typically dominates the runtime of the KSVD algorithm, the computational modifications presented in Section 4.1 allow us to significantly decrease the runtime to the point where it roughly matches that of the sparse encoding step.

3 Theoretical Aspects of Sparse Encoding and Dictionary Learning

Even following the hypothesis that transformer embeddings are superpositions of linear monosemantic features, it is not clear whether the dictionary learning problem is identifiable, i.e., whether there can exist an algorithm that can recover the true dictionary D and sparse assignments X solely from data samples Y. For this reason, it is useful to understand our problem in the context of the well-established theory of the dictionary learning problem and the sparse encoding subproblem. We use this theory to motivate the need for a scalable adaptation of the KSVD algorithm and to inform our algorithmic and experimental design choices.

3.1 The Ill-Posed Nature of Dictionary Learning

Without sparsity constraints on X, the dictionary learning problem from Eq. (1) is ill-posed in the sense that an infinite number of solutions (D, X) exist. This non-identifiability issue is a common challenge in unsupervised learning. For example, Locatello et al. (2020a) showed that finding monosemantic disentangled representations from observational data without appropriate inductive bias is impossible. However, for many types of data such as images or natural language, we can impose additional constraints on the dictionary learning problem. In particular, we can assume that each data sample is only composed of a few monosemantic features such that we can impose a sparsity constraint on the coefficient matrix X. This constraint transforms the problem from an underconstrained problem to a sparse dictionary learning problem, which is potentially identifiable.

However, even with the sparsity constraint, the dictionary learning problem is not guaranteed to be identifiable. Instead, the identifiability depends on problem parameters such as the dimensionality of the embeddings d, the sample size n, the number of dictionary elements m, the level of sparsity k, the "level of orthogonality" (incoherence) of the dictionary elements d_i, and the magnitude of noise or unmodeled terms. To understand the impact of these parameters in the context of our problem, we turn to the theoretical aspects of the dictionary learning and sparse encoding problems. For example, for low levels of sparsity (high k), just solving the simpler sparse encoding subproblem of finding x given D and y becomes difficult or infeasible. Furthermore, even if the sparse encoding problem is feasible for a fixed D, the simultaneous identification of D and X is even more challenging. The remainder of this section aims to build intuition for how the difficulty of the dictionary learning problem depends on the problem parameters.

3.2 The Identifiability of Sparse Encoding

The identifiability of the sparse encoding problem is fundamentally a function of the sparsity k and the incoherence of D. Intuitively, a large sparsity admits exponentially many more possible combinations of the columns of D. Furthermore, when the columns of D are near parallel, it is difficult to determine which columns were used to generate a given data sample y. Formally, the coherence of D is defined as $\mu(D) = \max_{i \neq j} |\langle d_i, d_j \rangle|$ and provides a measure of the orthogonality of the dictionary elements. We can relate the coherence of D to the maximum value for k such that the sparse vector x is unique and can be recovered through algorithms such as OMP or L_1-minimization.

Specifically, the condition

$$k < \frac{1}{2}(1 + \mu^{-1}(D)) \tag{5}$$

is a sufficient condition (Tropp & Gilbert, 2007, Thm. B) for recoverability. For k to be as large as possible, we want D to have minimal coherence, i.e., to have maximally orthogonal columns. We find that KSVD, by default, produces highly-coherent dictionaries, which motivates the extension described in Section 4.2. We note that coherence can be a "blunt instrument" because it only considers the maximum coherence, and the related Restricted Isometric Property (RIP) can be used as a more nuanced metric (Candès et al., 2006; Candès & Wakin, 2008; Foucart & Rauhut, 2013).

We can also use Eq. (5) to relate the number of recoverable features to the dimensionality of the samples. The coherence of maximally incoherent dictionaries is limited by the Welch bound (Welch, 1974) with $\mu(D) \geq \sqrt{(m-d)/(d(m-1))}$. If the number of dictionary elements m is a similar order of magnitude compared to the embedding dimension d, we can approximate the bound as $\mu(D) \geq \sqrt{1/d}$. Plugging this result into Eq. (5), we find that in the optimal case of maximally incoherent dictionaries the number of guaranteed identifiable elements scales with $\sqrt{d}$. This result indicates that the dimensionality d of the data must be sufficiently larger than the number of monosemantic features k in each sample.

3.3 Information-Theoretic Limits for Sample Complexity of Dictionary Learning

Even when the sparse encoding problem is identifiable, simultaneously identifying D and X is even more challenging and generally requires a large number of samples. This challenge is exacerbated further by the fact that $Y = DX$ is an imperfect model, and we instead have an unmodeled term ϵ such that $Y = DX + \epsilon$. Jung et al. (2016) take an information-theoretic approach to characterize the required number of samples for dictionary identifiability in this setting. They assume a Gaussian distribution for both the nonzero coefficients in X and the unmodeled term ϵ with variances σ_X^2 and σ_ϵ^2, respectively. The authors establish a lower bound on the required number of samples that is proportional to m^2 and inversely proportional to the signal-to-noise ratio SNR $= \sigma_X^2/\sigma_\epsilon^2$ (Jung et al., 2016, Eq. 21). Intuitively, this relationship indicates that (i) learning additional dictionary elements requires a quadratic number of additional samples and (ii) we can only identify nonzero elements if their signal is sufficiently large compared to the unmodeled term or "noise" ϵ.

3.4 Practical Considerations

Returning to our application of disentangling large transformer embeddings such as the embeddings from the Gemma-2-2B model (Mesnard et al., 2024), we propose the following practical considerations: **(i)** The large embedding dimension of transformer models ($d = 2304$ for Gemma-2-2B) benefits the number of recoverable monosemantic features k per sample. Conversely, if this method is applied to embeddings with a smaller dimension, the number of recoverable features may be smaller. **(ii)** The incoherence of the learned dictionary D is an important characteristic of the feasibility of the dictionary learning problem, and it can be used as a diagnostic tool to indicate dictionary performance on downstream metrics. **(iii)** Although the superposition hypothesis suggests a larger number of dictionary elements m than the data dimension d, we cannot arbitrarily increase m without simultaneously scaling the number of data samples n quadratically.

4 Double-Batch KSVD

This section outlines the main components of the DB-KSVD and Matryoshka DB-KSVD algorithms.

4.1 Scaling KSVD to Millions of Samples

Previous efforts have proposed a variety of algorithmic modifications to improve the performance of the KSVD algorithm on single-core (Rubinstein et al., 2008), multi-core (Sukkari et al., 2017), and hardware-accelerated systems (Irofti, 2014; He et al., 2016). However, these implementations have only been scaled to datasets with hundreds of dimensions and tens of thousands of samples, multiple orders of magnitude smaller than our requirements. In this section, we discuss algorithmic modifications to the classic KSVD algorithm that make it possible to scale it to thousands of dimensions and millions of samples. Specifically, we make modifications that allow us to reduce the computations needed in each step of the algorithm and map the problem to a scalable number of CPU workers. We also discuss how hardware acceleration can optionally be used to offload the most expensive operations, specifically those that scale with the number of samples n. Finally, we show how batching can be used to scale to even larger sample sizes beyond the limits of the working memory in a similar manner to how mini-batching is done for many machine learning workflows.

Parallel Matching Pursuit. The sparse encoding step of the KSVD algorithm initially appears to be "embarrassingly parallel" in the samples y_i such that it can be easily scaled to a large number of workers. However, when encoding a large number of samples with the same dictionary D, the performance can be enhanced significantly. From Eq. (3), we can see that $\langle r, d_j \rangle$ are elements of $D^\top r$, which we can store. Instead of recomputing $D^\top r$ after every update, we can notice the recurrence relation $D^\top r^{(t+1)} = D^\top r^{(t)} - \langle r, d_j \rangle (D^\top \, d_j)$. Further, we can fully avoid computing any matrix-vector products during the sparse encoding iterations by precomputing $D^\top D$, which is constant across all iterations and all samples y_i. For a more detailed explanation, we refer to Section 3.3 of Davis et al. (1997) where similar optimizations have been proposed. We can also initialize the product vector $D^\top r^{(0)}$ for each sample y_i by precomputing the matrix $D^\top Y$ and initializing each product vector with the corresponding column. With $D^\top D$ and $D^\top Y$ precomputed, each of the k iterations for each sample reduces to (i) an $\arg\max$ operation over the precomputed product vector $D^\top r^{(t)}$, (ii) indexing into $D^\top r^{(t)}$ to retrieve $\langle r, d_j \rangle$, and (iii) updating the product vector through a simple vector addition.

This optimized approach significantly shifts the computational burden. The precomputation of $D^\top D$ and $D^\top Y$ requires $O(dm^2)$ and $O(dmn)$ operations, respectively. In contrast, the subsequent k sparse encoding iterations for each of the n samples involve approximately $O((m + d)kn)$ operations. Since $k \ll d < m$, the cost of precomputing $D^\top D$ and $D^\top Y$ dominates the overall runtime of the entire sparse encoding problem when using MP. When heterogeneous compute is available, we can further speed up the computation by offloading the two matrix products to hardware accelerators such as GPUs. In practice, we find that this approach can be scaled efficiently to both high-CPU and heterogeneous CPU-GPU machines.

Inner Batching: Batched and Accelerated Dictionary Updates. The dictionary update step of the original KSVD algorithm has an inherently sequential nature. Specifically, in Eq. (4), when updating the jth dictionary element, we require the modified D and X matrices resulting from the previous dictionary element update. One way to circumvent this sequential nature and parallelize the dictionary update step is to use D and X of the previous KSVD iteration for all dictionary elements, rather than from the most recent update. However, this approach may deteriorate the convergence properties of the algorithm (Aharon et al., 2006, Sec. IV.C).

Instead, we propose to partition the m dictionary update tasks into shuffled batches of size w, where w is the number of available workers, e.g., CPU threads. Next, each worker performs one dictionary update on a local copy of the most recent dictionary D and sparse assignment matrix X. After all workers simultaneously perform their update, the updated column in D and row in X is synchronized with all other workers in an all-to-all fashion. Then, the next batch of w dictionary update tasks is considered and the process is repeated until all dictionary elements have been updated. We note that this adapted algorithm is not mathematically equivalent to the original KSVD algorithm. However, we find that in practice, the simultaneous batched updates do not significantly deteriorate convergence speed and allow highly efficient scaling of the dictionary update tasks to many CPU workers.

Next, we turn to the effect of the number of samples n on the computational cost of the dictionary update step. If we assume the nonzero elements in X are evenly distributed, the error matrix E_{Ω_j} will have approximately nk/m columns. As we increase n, computing the maximum singular vector of E_{Ω_j} can become a computational bottleneck. We address this bottleneck in three steps. First, we note that we can avoid computing the full singular value decomposition of E_{Ω_j} by using an iterative algorithm to compute only the maximum singular vector. Second, the singular vectors of E_{Ω_j} can be computed from the eigenvectors of either $E_{\Omega_j}^\top E_{\Omega_j}$ or $E_{\Omega_j} E_{\Omega_j}^\top$. For large n, we can therefore compute the maximum eigenvector of the symmetric matrix $E_{\Omega_j} E_{\Omega_j}^\top \in \mathbb{R}^{d \times d}$ with an iterative algorithm. We find that the Lanczos algorithm (Lanczos, 1950) performs best for this problem.

We note that the computational complexity of the eigenvalue problem no longer depends on the number of samples n. Further, computing $E_{\Omega_j} E_{\Omega_j}^\top$ can also be offloaded to hardware acceleration if available, providing significant additional speedup. However, to recover the singular vector from the computed eigenvector, a single multiplication with E_{Ω_j} is still necessary, so we cannot offload the entire dependence on n. Nonetheless, this optimization enables us to significantly reduce this dependence, allowing us to scale to problem sizes with millions of samples and to leverage both high-CPU and heterogeneous CPU-GPU machines.

Outer Batching: KSVD for Out-of-Core Problem Sizes. The algorithmic modifications to the sparse encoding and dictionary update step, combined with careful reduction of memory allocations, memory movement, and threading overhead, allow us to efficiently process up to one million samples at a time. However, for problems with more samples, the KSVD update step as written so far will exceed memory capacity. To remedy this problem, instead of using all available data at each DB-KSVD iteration, we propose batching the data in a similar manner to mini-batching for gradient descent-based training. In particular, at each step of DB-KSVD, we execute the sparse encoding and dictionary update steps on only a subset of the data samples. We find that this batching step is also beneficial for problem sizes that fit into the working memory. For a fixed time budget, the increased number of DB-KSVD iterations due to smaller batches outweighs the benefit of using the maximum number of samples in each step, provided that the batch size is large enough (typically one to two orders of magnitude larger than the data dimension d).

4.2 Imposing Inductive Bias with Matryoshka Structuring

As outlined in Section 3.2, incoherence is an important property of the learned dictionaries. However, as we present in Section 5.2, the dictionaries learned by DB-KSVD tend to have high coherence, especially for small values of k. We therefore propose additional structure as a regularization technique to encourage more incoherent dictionaries. A promising approach for imposing additional structure is the Matryoshka structuring, which was recently proposed in the context of SAEs (Bussmann et al., 2025) and representation learning (Kusupati et al., 2022). The Matryoshka structuring introduces a hierarchical ordering of dictionary elements that aims to encourage different layers of semantic detail.

We implement the Matryoshka structuring as follows. Following Bussmann et al. (2025), we partition the dictionary elements into groups of exponentially increasing size. Each group is assigned an equal share of the total budget of k nonzero elements. At each iteration of the KSVD algorithm, instead of doing a single sparse encoding and dictionary update step, we begin by performing these steps using the first group of dictionary elements and assignments only, which we refer to as $D^{(1)}$ and $X^{(1)}$. We then compute the residual matrix $E^{(1)} = Y - D^{(1)}X^{(1)}$ and fit the next group of dictionary elements to this residual matrix $E^{(1)}$. We repeat this process for all groups. While we perform the sparse encoding step on all subgroups separately during training, we find that a single sparse encoding step on the full dictionary is sufficient at evaluation time.

5 Experiments

To evaluate the quality of the dictionaries learned by DB-KSVD, we construct dictionaries from 2.6 million embeddings of the Gemma-2-2B and Pythia-160M models ($d = 2304$) evaluated on the Pile Uncopyrighted dataset (Gao et al., 2020). We learn dictionaries of size $m \in \{4096, 16384\}$ with sparsity $k \in \{20, 40, 80\}$ by running 40 iterations of batch size $n = 2^{16}$ (specific training details are summarized in Section A). These parameters are chosen because we notice no further performance improvements in our proxy metrics for more data or iterations (see Section B). We report timing results of our CPU-only and hardware accelerated implementations and baseline against an implementation of the KSVD algorithm as introduced by Aharon et al. (2006). Following the SAEBench benchmark (Karvonen et al., 2025), we compute six metrics on the resulting dictionaries comprising loss recovered, autointerpretability, absorption, sparse probing, spurious correlation removal, and attribute-value entanglement. We also study the coherence of the learned dictionaries and analyze the impact of induced bias from Matryoshka structuring. We conclude by demonstrating the benefits of initializing the dictionary with a partially known structure on a toy problem.

		CPU only	CPU+GPU	Baseline (est.)
$m = 4096$	$k = 20$	5.47	11.43	6.5×10^4
	$k = 40$	10.86	24.03	1.0×10^5
	$k = 80$	337.63	486.43	1.6×10^5
$m = 16\,384$	$k = 20$	17.12	23.23	7.4×10^4
	$k = 40$	21.51	31.77	1.3×10^5
	$k = 80$	41.95	72.79	2.6×10^5

Table 1: Runtime in seconds for a single batch of $n = 2^{16} = 65\,536$ Gemma-2-2B embeddings. We report the fastest of 5 trials, except baseline, which is estimated based on timing results from a smaller problem. All CPU only and CPU+GPU trials are within a 10 % margin.

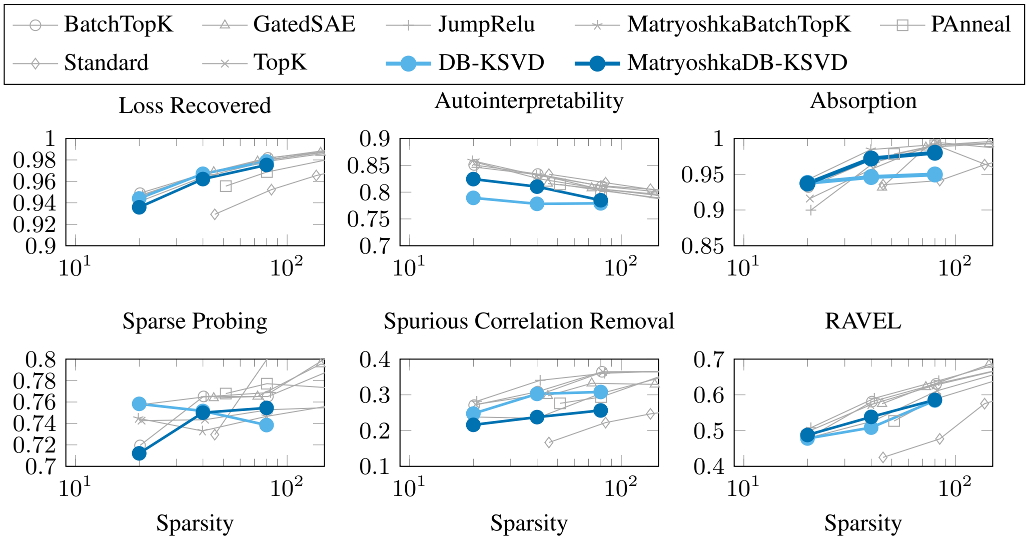

Figure 1: Results (higher is better) of our DB-KSVD algorithm and Matryoshka adaptation on the SAEBench benchmark appied to Gemma-2-2B embeddings with 4096 dictionary elements.

5.1 Timing Results

Table 1 shows the runtime results for one DB-KSVD iteration for different problem sizes and compute types. The CPU only results are generated on a Google Cloud Platform n2-standard-128 machine, and the hardware accelerated (CPU+GPU) results are generated on a n1-standard-96 machine with an NVIDIA T4 GPU. DB-KSVD is up to 10 000 times faster than the baseline implementation of KSVD, which still has access to all CPU cores for matrix operations. This result highlights the importance of our algorithmic modifications for the scalability of the DB-KSVD algorithm. In practical terms, for one full run on the 2.6 million embeddings with 4096 dictionary elements and $k = 20$, the DB-KSVD algorithm takes approximately 8 minutes to converge, while the baseline would take over 30 days.

For this problem size, the hardware acceleration does not improve the runtime. Results for larger problem sizes are presented in Section B. For $m = 4096$, the runtime drastically increases when increasing k from 40 to 80 for both compute types. However, this trend does not appear for $m = 16\,384$. We explain this increase by noticing that for a fixed k, a smaller m will result in a more dense X and therefore a wider E_{Ω_j} matrix. This increased width shifts the computational bottleneck, resulting in a larger runtime.

5.2 SAEBench Results

We evaluate the performance of our trained dictionaries from the DB-KSVD algorithm and the Matryoshka adaptation on the SAEBench benchmark and present the results for Gemma-2-2B with

$m = 4096$ in Fig. 1. Results for $m = 16\,384$ as well as for Pythia-160M are provided in Section B and show similar trends. The dictionaries learned using DB-KSVD outperform the Standard ReLU SAE approach (Cunningham et al., 2023; Bricken et al., 2023) in all metrics except autointerpretability and one instance of sparse probing. Moreover, the results are also generally competitive with all other SAE variants including the MatryoshkaBatchTopK SAE (Bussmann et al., 2025), except in the case of autointerpretability. The fact that two completely different optimization approaches (DB-KSVD and SAEs) achieve similar performance results may indicate that we are close to the theoretical limits given the problem size.

We hypothesize that the differences in autointerpretability are related to the coherence of the learned dictionaries, which we analyze in the next section. We also notice that the Matryoshka structuring approach improves autointerpretability and absorption performance; however, it slightly degrades the performance on the loss recovered and SCR metrics. These trends are similar to the trends observed when using the SAEs with Matryoshka structuring, hinting at a more fundamental phenomenon.

5.3 Coherence Metrics

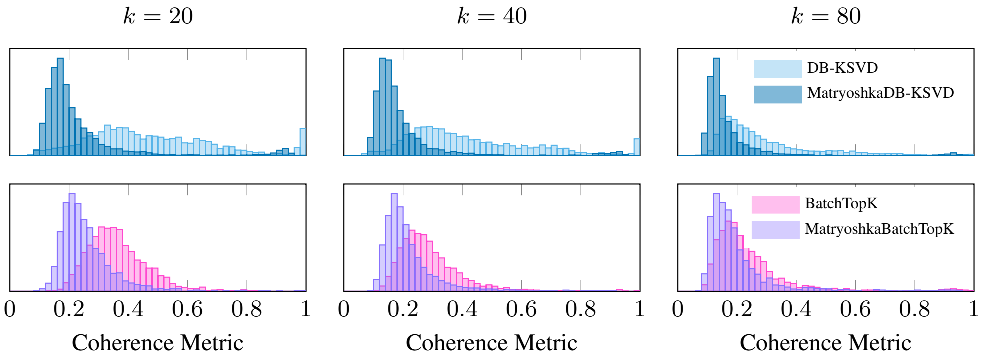

Figure 2: Histograms of element-wise coherence metrics for different sparsities. The dictionaries are constructed using the Gemma-2-2B embeddings and comprise $m = 4096$ dictionary elements.

To further understand these metrics, we recall from Section 3 that dictionary coherence is an important property for the well-posedness of the sparse encoding problem. We study the coherence of the learned dictionaries by computing $\max_{\ell \neq j} \langle d_j, d_\ell \rangle$ for each dictionary element d_j and show the results for varying sparsity levels k in Fig. 2. We also compare to SAE dictionaries constructed by Karvonen et al. (2025) for the same problem. We find that the learned dictionaries for DB-KSVD are relatively coherent, especially for smaller values of k. Additionally, there is a notable spike in dictionary elements that are almost perfectly aligned, i.e., $\langle d_j, d_\ell \rangle \approx 1$. This result motivates the extension introduced in Section 4.2 to encourage more incoherent dictionaries. When using the Matryoshka structuring approach, we indeed observe a significant decrease in the coherence of the learned dictionaries. We observe a similar trend when the Matryoshka structuring is introduced to the SAE approach. Furthermore, we hypothesize that the improvement in dictionary incoherence relates to the improved autointerpretability of MatryoshkaDB-KSVD over DB-KSVD. However, it is unclear whether this result is due to improved sparse encoding performance or the dictionaries themselves.

5.4 Initializing With a Known Structure

Finally, we investigate an advantageous property of AO algorithms: we can control the initialization of both D and X if something about them is known. Previous work in disentanglement and representation learning has emphasized the benefit of incorporating additional knowledge about the structure of the problem (Locatello et al., 2020b; Zbontar et al., 2021; Bardes et al., 2022). Motivated by this work, we study the effect of initializing with a partially known structure of X on a toy problem.

To set up the toy problem, we construct a randomly sampled dictionary D and a sparse coefficient matrix X with 20 nonzero elements per column ($d = 256$, $m = 1024$, $n = 2^{14}$, further details in Section A). We generate the data Y according to the model $Y = DX + \epsilon$, where ϵ is a Gaussian noise term with variance 0.05^2. For each experiment, we initialize X with the true adjacencies (0 or 1)

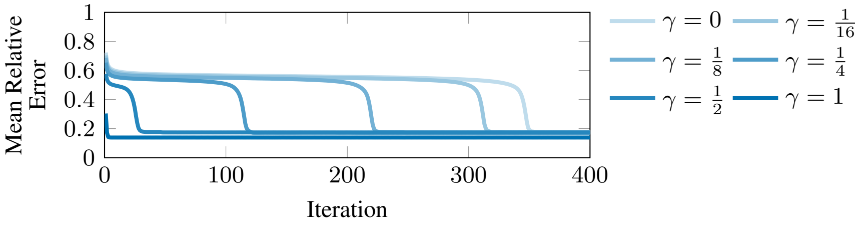

Figure 3: Convergence of the DB-KSVD algorithm with varying fractions of known adjacencies γ. We plot the mean relative error at each iteration, defined as $\frac{1}{n}\sum_i \|y_i - Dx_i\|_2 / \|y_i\|_2$.

of a fraction γ of the dictionary elements and initialize the remaining elements to zero. We run the DB-KSVD algorithm for 400 iterations and compute the average relative error at each iteration.

The results are shown in Fig. 3. Every experiment with $\gamma \neq 1$ converges to the same loss minimum. However, the number of iterations required to reach the minimum decreases significantly as γ increases. For instance, knowing $1/4$ of the true structure decreases the required iterations by $2/3$. We also observe a loss gap between $\gamma = 1$ and all $\gamma \neq 1$, which indicates that the true adjacencies have not been perfectly identified. Based on these results, we conclude that further investigation into using known structures to improve performance is a promising direction for future work.

6 Related Work on Interpreting Transformer Models

The challenge of interpreting large language models and other transformer architectures has prompted significant research. Early efforts focused on probing, where supervised methods are used to determine if predefined concepts have linear representations in model embeddings (Li et al., 2023; Nanda et al., 2023). While effective for verifying known features, probing does not readily uncover novel concepts learned by the model. A prevailing hypothesis for unsupervised concept discovery is that LLM activations represent many distinct features in a superimposed, entangled manner (Elhage et al., 2022). To disentangle these features, dictionary learning approaches aim to find a basis of monosemantic features. SAEs have emerged as a prominent and scalable technique for this task (Bricken et al., 2023; Templeton et al., 2024; Cunningham et al., 2023). Numerous SAE variants have been proposed to enhance performance, including methods like TopK activations to remove the need for L_1 penalty tuning (Gao et al., 2024) and Matryoshka representations (Bussmann et al., 2025) to introduce beneficial inductive biases. This rapid development led to the creation of SAEBench (Karvonen et al., 2025), a standardized benchmark for comparing learned dictionaries. The insights gained from disentangled representations are valuable for downstream applications such as understanding circuit-level computations (Lindsey et al., 2025; Ameisen et al., 2025), tracking feature dynamics (Xu et al., 2024), steering model behavior (Zhao et al., 2025), and enabling sparse routing mechanisms (Shi et al., 2025).

7 Conclusion

We have shown that AO algorithms can be scaled to problem sizes relevant for mechanistic interpetability of large transformer models. We achieve competitive results to state-of-the-art SAE approaches evaluated on a variety of standardized metrics for multiple models, which may indicate that both methods approach the theoretical limits of these metrics. We also showed that by focusing on coherence, a property well-studied in the context of sparse dictionary learning, we can explain trends in the autointerpretability and absorption properties. Although DB-KSVD can be scaled to millions of samples, it is not clear whether this sample size is enough or whether more scaling is needed to solve the disentanglement problem. Furthermore, DB-KSVD is algorithmically more complex than SAE approaches, which makes implementation more challenging. Ultimately, we have provided a new perspective on the disentanglement problem and hope that our implementation enables a wide range of applications of the KSVD algorithm.

REFERENCES

Alekh Agarwal, Animashree Anandkumar, Prateek Jain, and Praneeth Netrapalli. Learning sparsely used overcomplete dictionaries via alternating minimization. *SIAM Journal on Optimization*, 26 (4):2775–2799, 2016.

M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 2025.

Adrien Bardes, Jean Ponce, and Yann LeCun. VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning. In *International Conference on Learning Representations*, 2022.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Bart Bussmann, Noa Nabeshima, Adam Karvonen, and Neel Nanda. Learning multi-level features with matryoshka sparse autoencoders. *arXiv preprint arXiv:2503.17547*, 2025.

Emmanuel J. Candès and Michael B. Wakin. An Introduction To Compressive Sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, March 2008.

Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006.

Hoagy Cunningham, Aidan Ewart, Logan Riggs Smith, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *International Conference on Learning Representations*, 2023.

Hugo Dalla-Torre, Liam Gonzalez, Javier Mendoza-Revilla, Nicolas Lopez Carranza, Adam Henryk Grzywaczewski, Francesco Oteri, Christian Dallago, Evan Trop, Bernardo P. de Almeida, Hassan Sirelkhatim, Guillaume Richard, Marcin Skwark, Karim Beguir, Marie Lopez, and Thomas Pierrot. Nucleotide Transformer: building and evaluating robust foundation models for human genomics. *Nature Methods*, 22(2):287–297, February 2025.

G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, 13(1):57–98, March 1997.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022.

Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Applied and Numerical Harmonic Analysis. Springer, 2013.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024.

Lu He, Timothy Miskell, Rui Liu, Hengyong Yu, Huijuan Xu, and Yan Luo. Scalable 2D K-SVD parallel algorithm for dictionary learning on GPUs. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16, pp. 11–18, New York, NY, USA, May 2016. Association for Computing Machinery.

Paul Irofti. Efficient gpu implementation for single block orthogonal dictionary learning. *arXiv preprint arXiv:1412.4944*, 2014.

Alexander Jung, Yonina C. Eldar, and Norbert Görtz. On the Minimax Risk of Dictionary Learning. *IEEE Transactions on Information Theory*, 62(3):1501–1515, March 2016.

Adam Karvonen, Can Rager, Johnny Lin, Curt Tigges, Joseph Bloom, David Chanin, Yeu-Tong Lau, Eoin Farrell, Callum McDougall, Kola Ayonrinde, et al. Saebench: A comprehensive benchmark for sparse autoencoders in language model interpretability. *arXiv preprint arXiv:2503.09532*, 2025.

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. *Advances in Neural Information Processing Systems*, 35: 30233–30249, December 2022.

C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45(4):255, October 1950.

Rasmus Munk Larsen. Lanczos bidiagonalization with partial reorthogonalization. *DAIMI Report Series*, (537), 1998.

Richard B Lehoucq, Danny C Sorensen, and Chao Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. 2023.

Christian Marius Lillelund, Henrik Bagger Jensen, and Christian Fischer Pedersen. Cloud K-SVD for Image Denoising. *SN Computer Science*, 3(2):151, February 2022.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025.

Yuan Liu, Stéphane Canu, Paul Honeine, and Su Ruan. Mixed Integer Programming For Sparse Coding: Application to Image Denoising. *IEEE Transactions on Computational Imaging*, 5(3): 354–365, 2019.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. A sober look at the unsupervised learning of disentangled representations and their evaluation. *Journal of Machine Learning Research*, 21(1):209:8629–209:8690, 2020a.

Francesco Locatello, Ben Poole, Gunnar Raetsch, Bernhard Schölkopf, Olivier Bachem, and Michael Tschannen. Weakly-Supervised Disentanglement Without Compromises. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 6348–6359. PMLR, November 2020b.

S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent Linear Representations in World Models of Self-Supervised Sequence Models. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 16–30. Association for Computational Linguistics, 2023.

Balas Kausik Natarajan. Sparse Approximate Solutions to Linear Systems. *SIAM Journal on Computing*, 24(2):227–234, April 1995.

Andreas Noack. TSVD.jl: Truncated singular value decomposition with partial reorthogonalization . `https://github.com/JuliaLinearAlgebra/TSVD.jl`, 2019.

Vishal M. Patel and Rama Chellappa. Sparse Representations, Compressive Sensing and dictionaries for pattern recognition. In *The First Asian Conference on Pattern Recognition*, pp. 325–329, November 2011.

Y.C. Pati, R. Rezaiifar, and P.S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pp. 40–44 vol.1, 1993.

Meisam Razaviyayn, Hung-Wei Tseng, and Zhi-Quan Luo. Dictionary learning for sparse representation: Complexity and algorithms. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 5247–5251, May 2014.

Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit. *Cs Technion*, 40(8):1–15, 2008.

Wei Shi, Sihang Li, Tao Liang, Mingyang Wan, Guojun Ma, Xiang Wang, and Xiangnan He. Route sparse autoencoder to interpret large language models. *arXiv preprint arXiv:2503.08200*, 2025.

Daniel A Spielman, Huan Wang, and John Wright. Exact recovery of sparsely-used dictionaries. *Conference on Learning Theory*, 2012.

Harmen Taeke Stoppels and Lauri Nyman. ArnoldiMethod.jl: Arnoldi Method with Krylov-Schur restart, natively in Julia. `https://github.com/JuliaLinearAlgebra/ArnoldiMethod.jl`, 2024.

Dalal Sukkari, Hatem Ltaief, Drew Schmidr, and Eduardo Gonzalez F. ecrc/ksvd: The KAUST SVD (KSVD) is a high performance software framework for computing a dense SVD on distributed-memory manycore systems. 2017. Publisher: Github.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024.

Robert Tibshirani. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Ivana Tošić and Pascal Frossard. Dictionary Learning. *IEEE Signal Processing Magazine*, 28(2): 27–38, March 2011.

Joel A. Tropp and Anna C. Gilbert. Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, December 2007.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

L. Welch. Lower bounds on the maximum cross correlation of signals (Corresp.). *IEEE Transactions on Information Theory*, 20(3):397–399, 1974.

Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in Time Series: A Survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023*, pp. 6778–6786, August 2023.

Yang Xu, Yi Wang, and Hao Wang. Tracking the feature dynamics in llm training: A mechanistic study. *arXiv preprint arXiv:2412.17626*, 2024.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stephane Deny. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 12310–12320. PMLR, July 2021.

Zheng Zhang, Yong Xu, Jian Yang, Xuelong Li, and David Zhang. A Survey of Sparse Representation: Algorithms and Applications. *IEEE Access*, 3:490–530, 2015.

Yu Zhao, Alessio Devoto, Giwon Hong, Xiaotang Du, Aryo Pradipta Gema, Hongru Wang, Xuanli He, Kam-Fai Wong, and Pasquale Minervini. Steering Knowledge Selection Behaviours in LLMs via SAE-Based Representation Engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5117–5136. Association for Computational Linguistics, April 2025.

A Experiment Details

A.1 SAEBench

For the SAEBench benchmark in Section 5 we construct dictionaries using embeddings of the Gemma-2-2B model (Mesnard et al., 2024) evaluated on the Pile Uncopyrighted dataset (Gao et al., 2020). The dictionaries are constructed on a Google Cloud Platform n2-standard-128 machine with 128 simultaneous workers. We learn dictionaries of size $m \in \{4096, 16\,384\}$ with sparsity $k \in \{20, 40, 80\}$ by running 40 iterations of batch size $n = 2^{16}$, thereby using $40 \times 2^{16} = 2\,621\,440$ unique samples. Each sample has dimension $d = 2304$ and is collected as the activations of the 12th layer of the Gemma-2-2B model. Because shuffling the data is difficult, each sample is first stored in a large buffer and then gradually written to a file in a shuffled order.

For the DB-KSVD algorithm we use regular Matching Pursuit for the sparse encoding which terminates just before the $(k+1)$th adjacency would be added. During the dictionary update we use an implementation of the Arnoldi Method Stoppels & Nyman (2024) with a low tolerance to compute the maximum singular vectors, although we have also observed good results with TSVD Noack (2019); Larsen (1998) or other types of Krylov subspace methods. Arpack (Lehoucq et al., 1998) can be used for validation but does not work well when called from multiple workers. When updating the dictionary elements, the order of their updates is shuffled every time. To initialize the dictionary for DB-KSVD we sample each dictionary index from a uniform distribution $U(-1/2, 1/2)$ and normalize each column. For Matryoshka DB-KSVD we use group sizes $\{256, 256, 512, 1024, 2048\}$ for the $m = 4096$ case and group sizes $\{256, 256, 512, 1024, 2048, 4096, 8192\}$ for the $m = 16\,384$ case.

During training we measure two proxy metrics: mean relative error

$$\frac{1}{n} \sum_i \frac{\|y_i - Dx_i\|_2}{\|y_i\|_2} \tag{6}$$

and variance explained

$$1 - \frac{1}{d} \sum_j \frac{\text{var}((Y - DX)_j)}{\text{var}(Y_j)} \tag{7}$$

where j indexes matrix rows. We plot results for both metrics in Section B.4.

A.2 Initializing With a Known Structure

To set up the toy problem, we construct a randomly sampled dictionary D and a sparse coefficient matrix X with $k = 20$ nonzero elements per column, and $d = 256$, $m = 1024$, $n = 2^{14}$. Each nonzero element in X is sampled from the uniform distribution $U(1, 2)$. The dictionary is initialized similarly to Section A.1.

B Additional Results

B.1 Additional Timing Results

In Table 2 we present runtime results for one DB-KSVD iteration on a very large batch size of $n = 2^{20}$ samples, complementary to the timing results with a batch size of $n = 2^{16}$ samples presented in Section 5.1. Besides the batch size, the results were obtained in a similar fashion to Section 5.1, except that only 48 workers were used simultaneously on the CPU+GPU machine to limit GPU memory use.

Although in Section 5.1 the introduction of hardware acceleration did not benefit the runtime results, for a large batch size the CPU+GPU approach outperforms the CPU only approach by a factor of up to 10 for $m = 4096$ and up to 5 for $m = 16\,384$. In practical terms, computing dictionaries with 40 iterations using this batch size and the CPU+GPU approach would take approximately 3 h 15 min for $m = 4096$ or 7 h 15 min for $m = 16\,384$.

		CPU only	CPU+GPU	Baseline (est.)
$m = 4096$	$k = 20$	1428	185	1.0×10^6
	$k = 40$	2797	292	1.6×10^6
	$k = 80$	5495	536	2.7×10^6
$m = 16\,384$	$k = 20$	433	313	1.2×10^6
	$k = 40$	2779	649	2.1×10^6
	$k = 80$	6156	1243	4.2×10^6

Table 2: Runtime in seconds for a single batch of $n = 2^{20} = 1\,048\,576$ samples of the Gemma-2-2B embeddings. Baseline results are estimated based on timing results from a smaller problem.

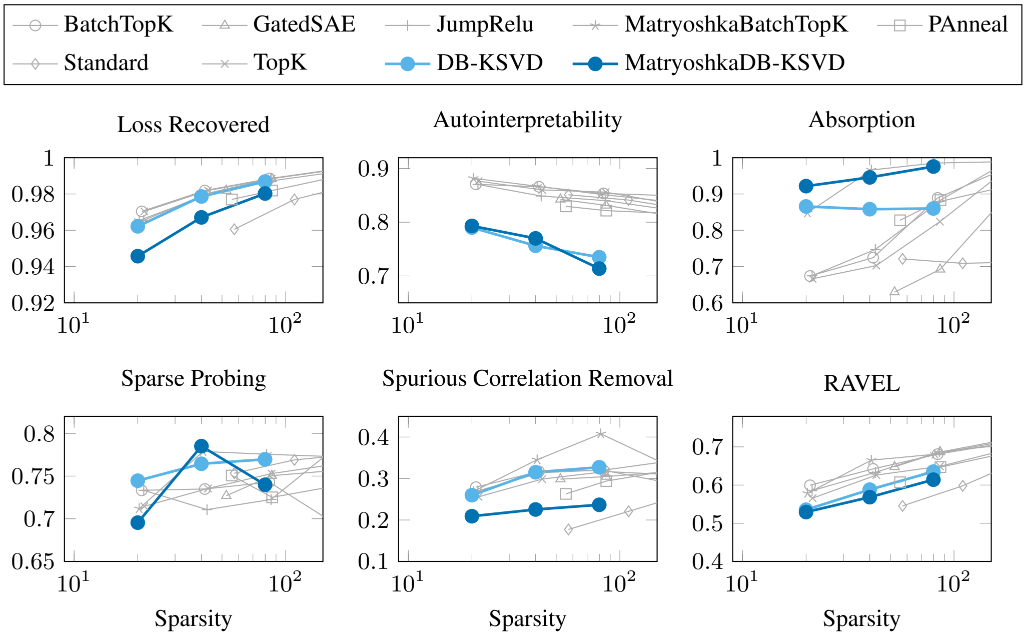

Figure 4: Results (higher is better) of our DB-KSVD algorithm and Matryoshka adaptation on the SAEBench benchmark applied to Gemma-2-2B embeddings with 16 384 dictionary elements.

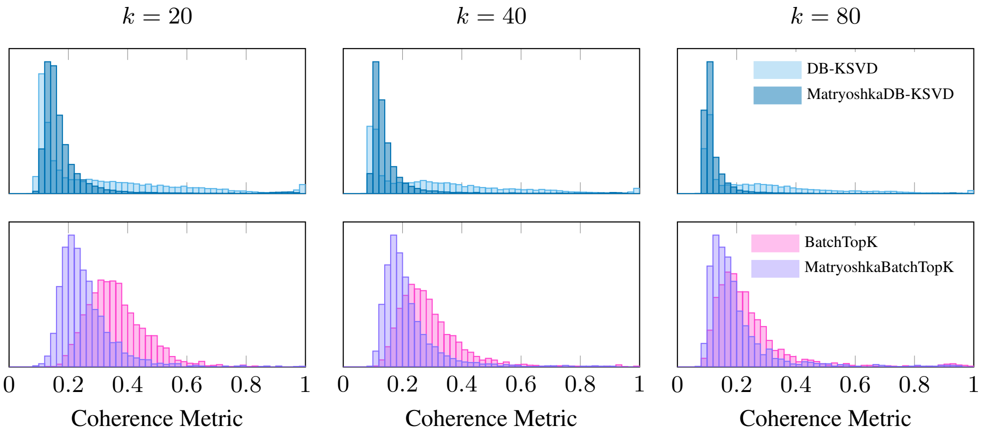

Figure 5: Histograms of element-wise coherence metrics for different sparsities. The dictionaries are constructed using the Gemma-2-2B embeddings and comprise $m = 16\,384$ dictionary elements.

B.2 SAEBench and Coherence Results for 16 384 Dictionary Elements

Figure 4 shows the performance of our trained dictionaries on the SAEBench benchmark similar to Section 5.2 but for $m = 16\,384$ dictionary elements. We observe similar trends as before for the loss recovered, sparse probing, spurious correlation removal and RAVEL metrics. However, unlike for $m = 4096$ the autointerpretability performance of Matryoshka DB-KSVD does not significantly improve compared to DB-KSVD in this case. However, the absorption metric of both methods is much improved relative to almost all SAE-based results.

Figure 5 shows the coherence metric of our dictionaries and comparable SAE based dictionaries similar to Section 5.3 for $m = 16\,384$. The results look similar to Fig. 2 except that the DB-KSVD results are relatively incoherent even without the Matryoshka modification.

B.3 SAEBench Pythia-160M Results

Figure 6 shows the performance of our trained dictionaries on the SAEBench benchmark similar to Section 5.2 but using embeddings from the Pythia-160M model. We were unable to find data files for the SAEBench baselines for the RAVEL metric with this model, so do not show comparisons for this metric. For the other metrics, we observe similar trends to the results we get using the Gemma-2-2B embeddings. These results indicate that our conclusions about the performance of DB-KSVD extend beyond a single model.

B.4 Proxy Metrics

Figure 7 and Fig. 8 display the mean relative error (Eq. (6)) and variance explained (Eq. (7)) proxy metrics that we use to asses model performance during training. At each step, both metrics are computed for the current training batch and a fixed and held out validation batch. We can see that both metrics have mostly flattened out after 40 iterations, which motivates our training setup discussed in Section 5.

C LLM Usage

We used large language models (LLMs) in a limited capacity during the preparation of this work. Specifically, LLMs were used to refine the phrasing and clarity of already-formulated sentences and as one component of finding potentially relevant literature to our ongoing work. All related work identified through LLM assistance underwent thorough manual verification and evaluation before inclusion in this paper.

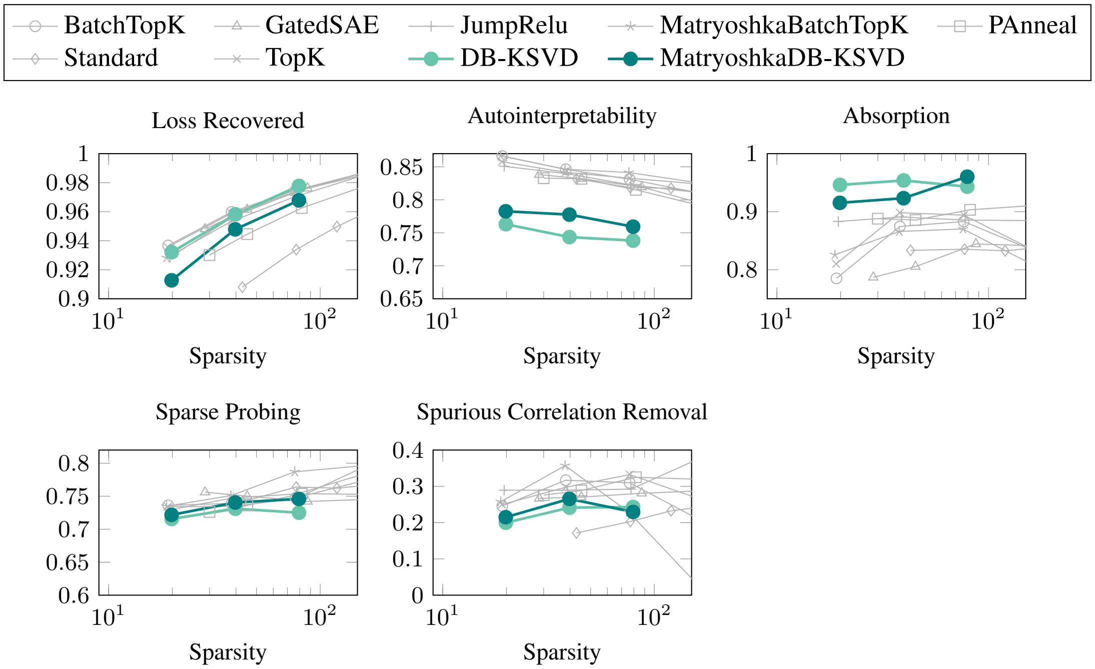

Figure 6: Results (higher is better) of our DB-KSVD algorithm and Matryoshka adaptation on the SAEBench benchmark with 4096 dictionary elements on Pythia-160M embeddings.

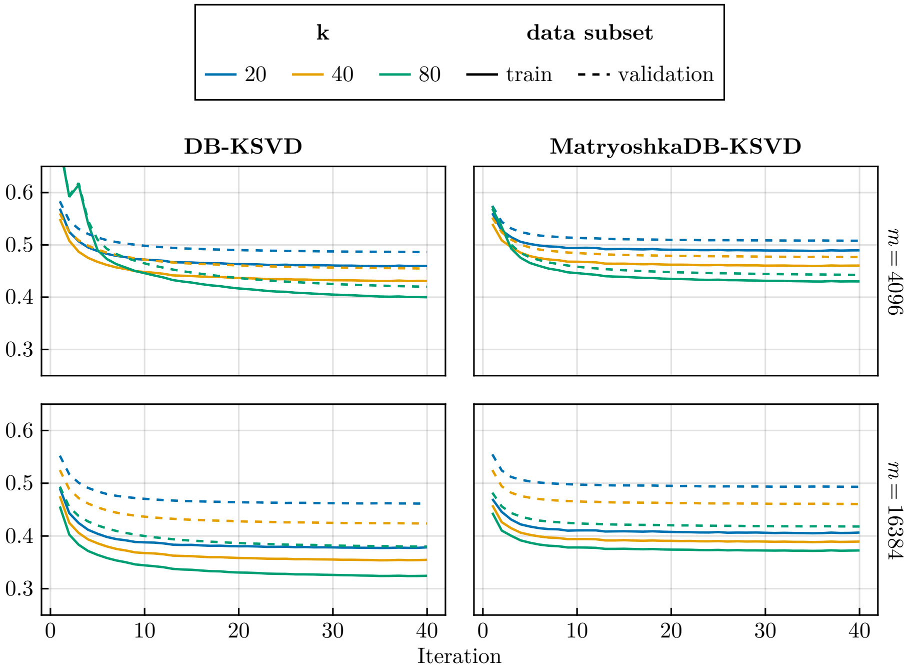

Figure 7: Mean relative error (Eq. (6)) for SAEBench training (Section 5) on the Gemma-2-2B model plotted against iterations.

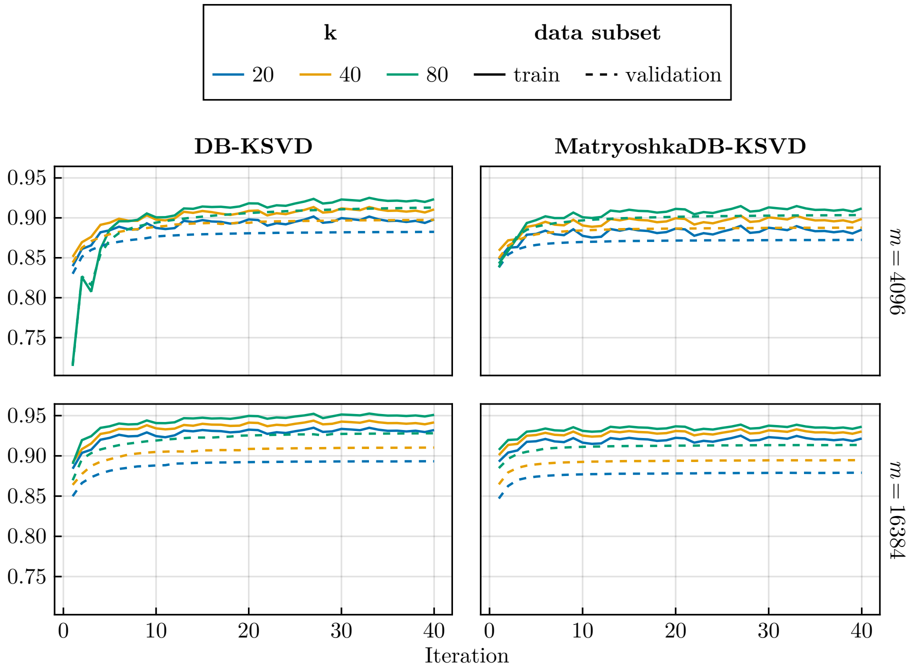

Figure 8: Variance explained (Eq. (7)) for SAEBench training (Section 5) on the Gemma-2-2B model plotted against iterations.