Finding Manifolds With Bilinear Autoencoders

Thomas Dooms
University of Antwerp
Antwerp, Belgium
doomsthomas@gmail.com

Ward Gauderis
Vrije Universiteit Brussel
Brussels, Belgium
ward.gauderis@vub.be

Abstract

Sparse autoencoders are a standard tool for uncovering interpretable latent representations in neural networks. Yet, their interpretation depends on the inputs, making their isolated study incomplete. Polynomials offer a solution; they serve as algebraic primitives that can be analysed without reference to input and can describe structures ranging from linear concepts to complicated manifolds. This work uses bilinear autoencoders to efficiently decompose representations into quadratic polynomials. We discuss improvements that induce importance ordering, clustering, and activation sparsity. This is an initial step toward nonlinear yet analysable latents through their algebraic properties.

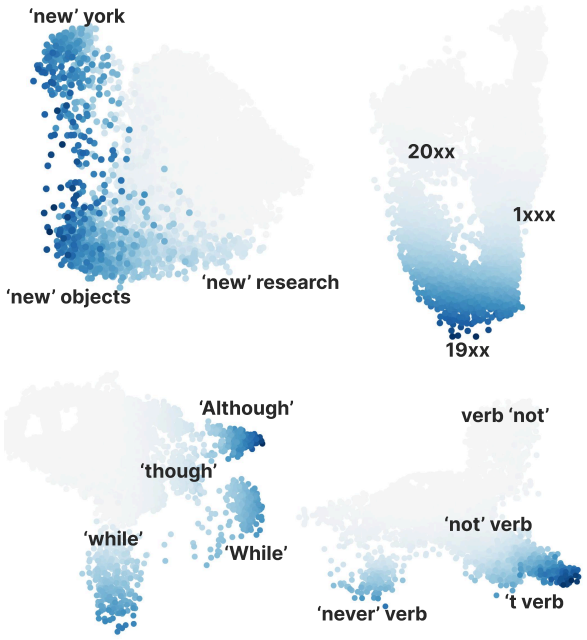

Figure 1: Conceptual subspaces extracted directly and solely from the weights of a bilinear autoencoder. To visualise the structure of these spaces, inputs are projected linearly onto it, with colour indicating activation strength. This reveals learned manifolds: the top-left organises nuances of 'new' into an off-centre triangle, while the top-right forms an off-centre circle for the first digits in a year. The bottom-left linearly clusters subordinating conjunctions, and the bottom-right captures verb negation. The top two are nonlinear manifolds, whereas the bottom two suggest linear concepts, possibly in superposition [13]. An interactive version is available at `https://mi-manifold.netlify.app/`.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Mechanistic Interpretability Workshop.

1 Introduction

Neural networks are highly nonlinear systems shaped by emergent phenomena rather than explicit mechanisms. Yet, human intuition is grounded in explicit structure and predictable composition. Mechanistic interpretability attempts to bridge this gap by identifying interpretable bases [24, 9], enabling latents to be (roughly) examined in isolation, despite being nonlinear [1]. Yet, the core problem remains: these latents lack a compositional interpretation — their isolated behaviour cannot be related to that of the whole [8]. Extracted latents have implicit domains of applicability: regions in input or activation space within which they are meaningful. For instance, latent splitting and absorption [7] split a latent's domain to increase sparsity while retaining its function. Consequently, their composition has a different meaning: instead of combining independent features, their domain is extended. As a result, the isolated study of latents is of limited value, since their composition may skew their interpretation, even at tiny scales [23]. Compositional understanding is vital for interpretability, demanding that dependencies and interactions be explicit and computable [33, 16].

One natural question is whether we can design or constrain nonlinear systems to remain compositionally interpretable [33]. Polynomials offer a solution as they are nonlinear yet fully analysable through their coefficients, independent of their representation basis [12, 27, 16]. They also admit direct geometric interpretation: many familiar manifolds, such as hyperspheres, are described by polynomial equations. These properties make polynomials a promising foundation for studying nonlinear computation and representations, offering a glimpse into their learned geometries (Figure 1).

This paper investigates bilinear autoencoders [16], which efficiently decompose neural representations into polynomial latents. This enables the uncovering of manifolds in an automated manner. It also allows exact computation of metrics such as similarity and distance between or within autoencoders, without relying on input-dependent correlation analysis. In short, this work contributes the following:

Architecture. We give an introduction to bilinear autoencoders, outlining their conceptual motivation and an efficient implementation.

Extensions. We propose several new extensions that induce clustering, ordering and sparsity, avoiding post-hoc regularisation penalties.

Analysis. We use bilinear autoencoders to uncover interesting nonlinear manifolds and leverage their compositionality to quantify latent consistency.

2 Bilinear autoencoders

This manuscript assumes some familiarity with tensor notation and adopts the diagrammatic tensor notation from [16] (for a short introduction, see [32]). We denote scalars (0-order tensors) as s (lower-case), vectors (1-order tensors) as $\mathbf{v}$ (bold lower-case), matrices (2-order tensors) as M (upper-case) and higher-order tensors as $\mathbf{T}$ (bold upper-case). We represent indexed tensors as $\mathbf{T}_{ijk}$, and denote slicing along tensor dimensions as $\mathbf{T}_{m:n}$ to extract subtensors.

2.1 Motivation and architecture

Autoencoders reconstruct their inputs through a bottleneck, forcing the model to capture structure in the data [18, 17]. They recently became a key tool in interpretability for extracting latents from neural representations [9, 4]. In this setting, the goal is to reconstruct and relate the discovered structure to the original representations, imposing a trade-off between latent expressivity and analysis complexity.

At the simplest level, linear decompositions such as singular value decomposition (SVD) can be considered autoencoders. However, low-rank constraints are often too restrictive: neural representations typically span all dimensions, so purely rank-based methods can fail to capture much of their structure. A more expressive alternative is the (sparse) autoencoder, which adds a nonlinearity while keeping a linear decoder. This allows recovery of linearly separable latents that remain directly traceable to the original representations. Both approaches aim to reveal structures that are easily extractable from the representation, but make assumptions about how those features are represented [17] As a result, they can produce latents that reflect the imposed prior rather than the underlying structure itself [14, 7].

[1]The code for this paper can be found at `https://github.com/tdooms/bae`.

Bilinear autoencoders [16] adapt sparse autoencoders to gated linear units [30, 11]. GLUs apply two matrices to their input $\mathbf{x} \in \mathbb{R}^{\text{In}}$ and multiply the output element-wise $\odot$, acting as a dynamic gate:

$$\text{GLU}(\mathbf{x}) \;=\; D\big(\sigma(L\mathbf{x}) \odot R\mathbf{x}\big).$$

The function σ can be removed with negligible impact on accuracy [27, 30]. The resulting operation can represent second-order polynomials of its inputs while remaining linearly analysable in the 'lifted' *product space* of all pairwise multiplicative interactions $\mathbf{x} \otimes \mathbf{x}$ between elements of $\mathbf{x}$. A single encoded *bilinear latent* $\mathbf{f}(\mathbf{x})_j$ ($j \in 1, \ldots, \text{Lat}$) is defined as.

$$\mathbf{f}(\mathbf{x})_j \;:=\; \mathbf{x}^\top \mathbf{l}_j \mathbf{r}_j^\top \mathbf{x} \;=\; (\mathbf{l}_j^\top \otimes \mathbf{r}_j^\top)\,(\mathbf{x} \otimes \mathbf{x}) \;:=\; B_j X$$

where $X = \mathbf{x} \otimes \mathbf{x}$ is the interaction vector (a 2-order tensor) and $B_j = (\mathbf{l}_j \otimes \mathbf{r}_j)^\top$ is a rank-1 matrix. Each bilinear latent thus corresponds to a rank-1 bilinear form. Stacking all B_j into a 3-order tensor yields the encoder $\mathbf{B}$ of shape $(\text{Lat} \times \text{In} \times \text{In})$. When $\mathbf{B}$ is reshaped into a $(\text{Lat} \times \text{In}^2)$ matrix, we denote it as B.

The decoder is the encoder's transpose, trained to reconstruct the original product representation. Untying these weights has been observed to have a negligible impact on performance metrics. Hence, the bilinear autoencoder approximates X as

$$\hat{X} \;=\; B^\top B X$$

Here, $\mathbf{f} := BX$ has fewer dimensions than $X = \mathbf{x} \otimes \mathbf{x}$ but more than $\mathbf{x}$, creating a (not necessarily sparse) bottleneck on the product space. This bottleneck corresponds to the amount of linearly extractable structure in the interaction matrix X, which is quadratic in $\mathbf{x}$.

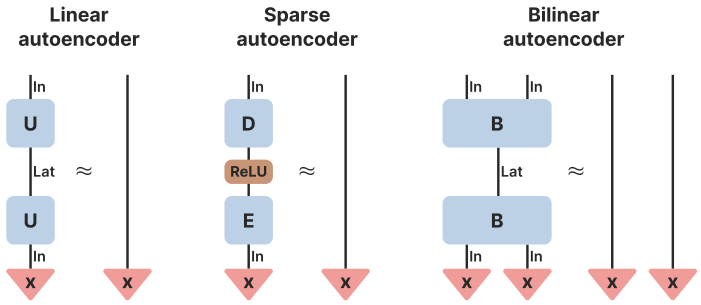

Figure 2: Most autoencoders reconstruct their inputs nonlinearly. Instead, bilinear autoencoders linearly reconstruct the product space X, which is quadratic in the input $\mathbf{x}$.

2.2 Bilinear latents and their geometry

The primary motivation behind current autoencoders [15, 29] is the linear representation hypothesis, claiming that most neural representations are encoded linearly [25, 9]. However, recent research suggests some salient concepts span more complicated manifolds, which we would also like to capture [22, 14]. Consequently, we want simple primitives which can be natively and meaningfully composed to encompass manifolds of varying complexity. Polynomials fit this role: latents capture linear directions in product space, and their closed-form bilinear composition yields quadratic manifolds in input space (Figure 3).

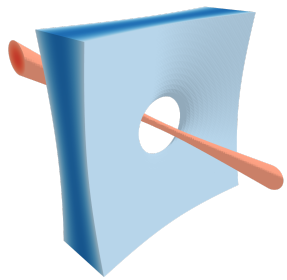

Figure 3: Coloured areas illustrate high activations $\mathbf{f}_i(\mathbf{x}) > 1$ for two bilinear latents B_i. Isolated latents can selectively activate on superposed directions (red). Composite latents can activate on more complicated manifolds, such as oriented cones and paraboloids (blue).

Single bilinear latents are similar to their ReLU-based counterparts in spirit; they activate strongly within cone-like regions. However, there are three subtle differences: bilinear latents are point symmetric across the origin, can have negative outputs, and do not have sharp activation boundaries. Despite the latter, latents still activate very selectively, as discussed in Appendix G.

Linear combinations of bilinear latents still have a closed form: they are represented by higher-rank bilinear forms. Each form defines an activation landscape, whose level sets manifest as quadratic surfaces. This enables combinations of latents to define intricate regions of high/low activation.

Bilinear latents' algebraic properties enable tractable analysis since they form a vector space; a single latent or any linear combination can be expressed as a bilinear form. This closure also yields basis independence, where rotations and mixtures of latents stay inside the same analysable vector space. This results in mathematical objects in which composite interactions can be made explicit and be manipulated, naturally extending interpretability from individual latents to their combinations [16].

2.3 Efficient training with the kernel trick

Evaluating bilinear autoencoders differs from ordinary ones since it reconstructs the quadratic space. While the inputs are naturally factored as $X = \mathbf{x} \otimes \mathbf{x}$, this is not true for the outputs $\hat{X} \neq \hat{\mathbf{x}} \otimes \hat{\mathbf{x}}$. Hence, naively computing the reconstruction loss $||\hat{X} - X||_F^2$ would require materialising this quadratic space. While this is feasible in small quantities, it becomes a significant hurdle for large-scale training. However, we can avoid this expensive computation by decomposing the sum of squares error (SSE; Appendix A) into terms.

$$\mathrm{SSE}(\hat{X}, X) = \sum_{i=1}^{\mathrm{In}} (B^\top B X - X)_i^2 = X^\top B^\top B B^\top B X - 2 X^\top B^\top B X + X^\top X \tag{1}$$

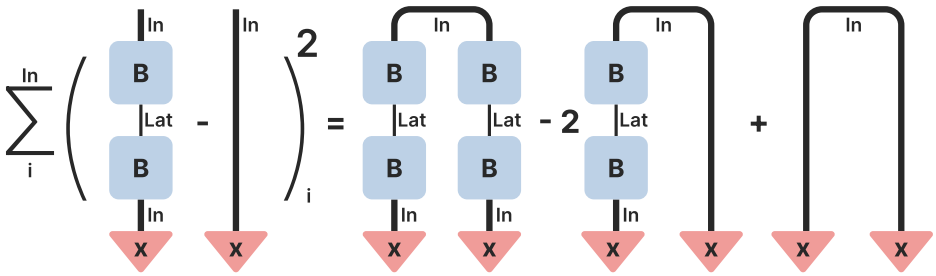

Figure 4: Diagrammatic formulation of Equation 1. Lines indicate tensor contractions over that index, while unconnected lines represent unmodified open indices. Thick lines represent a quadratic space.

Rewriting the latent representation BX as the vector $\mathbf{f}$, we get the following formula where $BB^\top$ is a matrix of shape (Lat $\times$ Lat).

$$\mathrm{SSE}(\hat{X}, X) \;=\; \mathbf{f}^\top (BB^\top)\,\mathbf{f} \;-\; 2\mathbf{f}^\top\mathbf{f} \;+\; X^\top X$$

This reframing avoids materialising the reconstruction. The latter two terms are inner products, and the first can be efficiently computed (Appendix B). This is reminiscent of the "kernel trick" for polynomials, which simulates an intractable feature space by computing efficient inner products using a similarity kernel. In this case, the inner product is parameterised. We refer to $BB^\top$ as the kernel matrix of the bilinear autoencoder.

In summary, evaluating bilinear autoencoders scales quadratically in time and linearly in memory with respect to the latent dimension. Despite the quadratic size of the kernel matrix, materialising it is not required. The term $\mathbf{f}^\top(BB^\top)\mathbf{f}$ is efficiently computable through contraction ordering and tiling to scale well (similar to flash attention [10] and face-splitting products). See Appendix B for details.

3 Extensions

Thus far, we have motivated and described a basic bilinear autoencoder. Here, we provide methods to efficiently induce additional structure into the learned latents. Specifically, we discuss activation sparsity, latent ordering and clustering. Note that these methods can apply to any autoencoder with a (multi-)linear decoder, but are not the focus of this present work.

3.1 Avoiding dead latents with scale-invariant sparsity

Activation sparsity is a good prior for finding monosemantic latents that correspond to a single concept. This sparsity is often enforced by minimising sparsity-inducing norms, like $L_{0<p\leq 1}$, of the activations. Yet, this penalty punishes absolute magnitudes, driving all activations to zero in its optimum. This trivial shrinkage of absolute scale is misaligned with our goal of promoting *relative* prominence. This can destabilise training by encouraging dead latents. We avoid this by using a scale-invariant measure, removing pressure toward shrinkage. Specifically, we use the Hoyer density measure [19], which smoothly measures the relative density of a latent $\mathbf{f}$.

$$\mathrm{Hoyer}_{\text{density}}(\mathbf{f}) \;=\; \frac{\frac{\|\mathbf{f}\|_1}{\|\mathbf{f}\|_2} - 1}{\sqrt{\mathrm{Lat}} - 1} \tag{2}$$

This measure ranges from 0 for one-hot vectors to 1 for perfectly uniform ones. Vectors with Gaussian entries score 0.8 in expectation, while a k-sparse vector yields at most k/Lat. We compute this value for each latent across all samples (batch $\times$ sequence). This batch-wise minimisation exerts dual pressure for latent variables to activate sharply but avoid collapse, producing selective yet robust latent representations [5]. By adding the Hoyer density as a penalty with coefficient α to the SSE loss, we obtain the following objective (where $\mathbf{f}_j$ are latent activations over a batch):

$$\mathcal{L}(\mathbf{x}) \;=\; \mathrm{SSE}(\hat{X}, X) \;+\; \alpha \sum_{j}^{\mathrm{Lat}} \mathrm{Hoyer}_{\text{density}}(\mathbf{f}_j), \tag{3}$$

3.2 Ordering latents by importance through cumulative reconstruction

Another desideratum for autoencoders is that they admit an explicit ordering with respect to feature importance, much like SVD. This allows the autoencoder to be truncated in (near-)optimal way to retain reconstruction quality. Currently, existing methods order through stochastic sampling or retraining on residual errors [6]. Ideally, we would like to impose total ordering during training. This is achieved by using a (weighted) cumulative reconstruction loss, promoting all prefix sets to achieve good reconstruction.

$$\mathrm{SSE}_{\text{cum}}(\hat{X}, X) \;=\; \sum_{i=1}^{\mathrm{In}} \sum_{k=1}^{\mathrm{Lat}} w_k \left(B^\top (BX)_{1:k} - X\right)_i^2 \;=\; \mathbf{f}^\top \big((BB^\top) \odot (M^\top \,\mathrm{Diag}(\mathbf{w}) M)\big)\mathbf{f} \;-\; \cdots \tag{4}$$

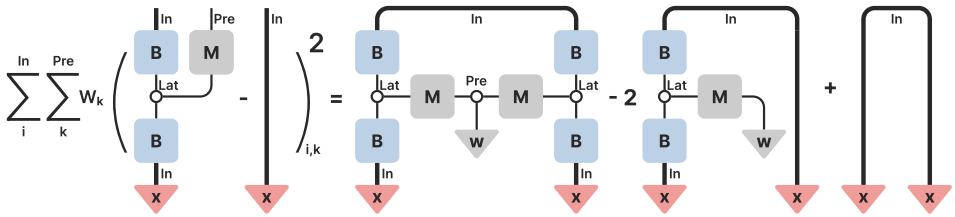

Figure 5: Diagrammatic formulation of the ordered loss. Here, M is an upper triangular mask matrix with columns $\mathbf{m}_k$ and $\mathbf{w}$ are the weights of each cumulative reconstruction. We use a uniform distribution for $\mathbf{w}$ to make all prefix reconstructions (Pre) equally important. This encourages the latents to contribute to the reconstruction in an approximately arithmetic progression of importance.

For each output element, we sum over all prefix reconstruction errors. This is achieved by multiplying the latents with a mask $\mathbf{m}_k = [\mathbf{1}_k, \mathbf{0}_{d-k}]^\top$. By collecting $\mathbf{m}_k$ as columns into a matrix, forming an upper-triangular mask matrix M, and collecting $\sqrt{w_k}$ into a vector $\mathbf{w}$, we can write

$$\sum_{k=1}^{\text{Lat}} w_k \big(B^\top(\mathbf{f} \odot \mathbf{m}_k)\big)_i^2 \;=\; \big(B^\top \mathbf{f} \otimes (M^\top \operatorname{Diag}(\mathbf{w}) M)\big)_i^2$$

Naively computing this suffers from the issues described in subsection 2.3: materialising these products is expensive. Hence, we perform the same trick and recast the ordered loss as a kernel matrix. Specifically, instead of $BB^\top$ we obtain $(BB^\top) \odot (M^\top \operatorname{Diag}(\mathbf{w}) M)$ which can be efficiently computed. The other terms left out in Equation 4 remain relatively unchanged, becoming a weighted inner product. Empirically, this yields autoencoders that reconstruct well with fewer latents (Appendix F).

3.3 Mixing latents through a linear bottleneck

Not all latents are one-dimensionally linear; some concepts span intricate multi-dimensional manifolds [14]. The challenge is finding latent combinations that likely contain meaningful structure. We introduce a bottleneck within the latent space, using a down-projection D into a lower-dimensional space, producing mixture latents whose bilinear form can attain higher rank.

$$\hat{X} \;=\; B^\top D^\top D\, B\, X$$

This is reminiscent of the setup of [13]: imposing the bottleneck encourages the model to identify (anti-)correlations among latents to improve reconstruction. In this way, the bottleneck highlights interacting multi-dimensional linear spaces that are likely to contain more complex manifolds.

3.4 Combining extensions

The extensions introduced above are modular and can be combined to produce autoencoder setups with different behaviour. In this paper, we apply ordering and sparsity to the latent basis. Alternatively, one could apply sparsity within the bottleneck from subsection 3.3, promoting sparse latent mixtures.

Combining these extensions may require attention to how penalties interact. In particular, ordering via cumulative reconstruction amplifies the contribution of early latents to the loss, making them more important for reconstruction. To preserve balance, other penalties such as activation sparsity should be scaled accordingly across the ordered basis.

$$\mathcal{L}(\mathbf{x}) \;=\; \text{SSE}(\hat{X}, X) \;+\; \alpha \sum_{k}^{\text{Lat}} \sum_{j}^{\text{Lat}} (\text{Lat} - k) \text{Hoyer}_{\text{density}}(\mathbf{f_j}) \tag{5}$$

4 Experiments

This section provides experimental evidence for the utility of bilinear autoencoders. We discuss their reconstruction and sparsity trade-off, how they help find manifolds, and their similarity across hyperparameters. We experimented with the following four different flavours of bilinear autoencoders.

- `vanilla`: contains only L and R matrices and uses Hoyer sparsity (subsection 3.1).
- `ordered`: adds a penalty to order latents by importance (subsection 3.2).
- `mixed`: adds an additional D matrix to cluster/mix latents (subsection 3.3).
- `combined`: a combination of mixed and ordered (subsection 3.4).

4.1 Measuring sparsity and reconstruction

Bilinear autoencoders differ from nonlinear variants in that they cannot learn hard cutoffs. Nevertheless, they achieve strong reconstruction, indicating that neural representations are well-separable within the product space. Due to their different reconstruction objective, direct comparison to sparse autoencoders is subtle. Appendix D shows that bilinear autoencoders can reconstruct the product space more effectively than sparse autoencoders.

Density–reconstruction analysis (Figure 6) shows that activation density (Equation 2) can be freely lowered from 0.6 to 0.2 (by increasing α from 0 to 0.1) at no cost to reconstruction, beyond which errors increase. While latent mixtures improve this trade-off, imposing an ordering degrades it. Intriguingly, density penalties sometimes improve reconstruction, suggesting that the model is learning naturally sparse features.

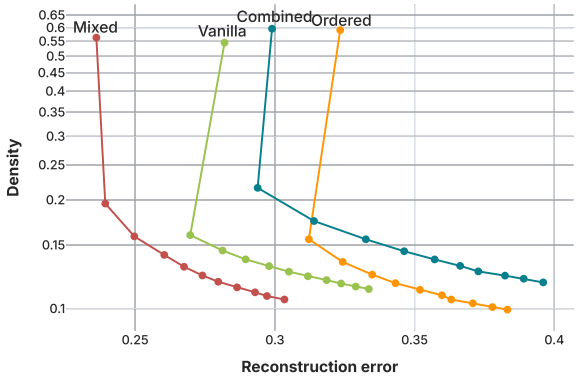

Figure 6: Reconstruction error (Equation 1) vs. activation density (Equation 2) across variants and sparsity penalties between $\alpha = 0$ (top) and $\alpha = 1$ (bottom) on layer 18 of Qwen3-0.6B. Lower error and density are better: activations can be strongly sparsified with minimal reconstruction trade-offs. The reconstruction is normalised between 0 and 1 (Appendix A).

The distribution of activation sparsities reveals further structure, especially when ordered (Figure 7). The `vanilla` and `ordered` variants exhibit one prominent peak of highly sparse latents and a smaller peak of dense latents. The ordering pushes the dense latents to the front, emphasising their importance [31]. Notably, this effect is independent of sparsity regularisation, as the penalties are adjusted (subsection 3.4).

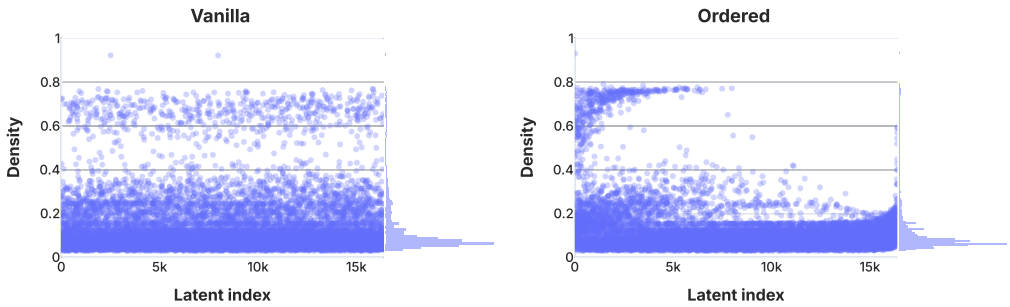

Figure 7: The Hoyer density of latents for the vanilla and ordered autoencoders (both $\alpha = 1$). This reveals an apparent dichotomy where about 2% of latents activate densely despite sparsification.

4.2 Discovering manifolds through latent interactions

Some features in language models live neatly along linear directions. Others occupy curved or folded manifolds that such linear views cannot capture [22]. To identify these manifolds, we train a `mixed` autoencoder (subsection 3.3) whose reconstruction passes through a constrained latent basis.

$$\mathbf{f}^\top D^\top D\mathbf{f} \;\approx\; I$$

The matrix $D^\top D$ encodes how latents co-behave. Clusters in this matrix signal latent groups that interact during reconstruction and are likely to contain multi-dimensional latent structure [13]. We quantify these strongly interacting latents by counting the number of 'high' elements in their rows. This is measured using the row's densities after some sharp function, in this case a fourth power: $\text{Hoyer}_{\text{density}}((D^\top D)_i^4)$. Similarly-sized outliers remain relevant while everything else is pushed to zero. The density measure is high if the original vector contains mostly similar-sized entries, while it will be low if it has a single outlier. We then consider the top 10 entries in our candidate cluster and combine them into a (weighted) composite latent.

Since this composite latent is a bilinear form, we can use its eigendecomposition to find the most salient 3D subspace in the original input space. Next, we define the activation strength of a composite latent as the L_2 norm of its individual activations. We then project the 64k highest activating inputs (out of 256k) onto the extracted subspace. The result is a fully linear slice of the input space, so any visible structure must be present in the original representation. In contrast to methods like PCA, this approach reproducibly reveals manifolds purely from the autoencoder's weights, without relying on input-dependent statistics.

The visualisations (Figure 1) reveal diverse geometries: smooth quadratic surfaces (top left and right), sharply separated clusters (bottom left), and linear directions (bottom right). We inspected about 50 candidates and picked the most interesting ones. Qualitatively, most concepts are linearly represented; their visualisation yields a large cloud of near-zero activations with one or a few directions sticking out with high activation. However, roughly half showed additional nonlinear structure. Such cases illustrate how the latent space can contain nonlinear manifolds beyond isolated directions.

4.3 Computing exact reconstruction similarity between autoencoders

How consistent are autoencoders? In most settings, the answer comes from partial evidence: a few encoder/decoder directions align [26], and reconstruction correlations are roughly the same. Bilinear autoencoders can give an exact weight-based answer to this question, without referencing a data distribution. Since they can be represented by a (matricised) 4-order tensor $A = B^\top B$, we can compare their Frobenius similarity directly.

$$\text{Similarity}_{\text{Frobenius}}(A, A') \;=\; 1 - \frac{\|A - A'\|_F^2}{\|A\|_F^2 + \|A'\|_F^2} = \frac{2\,\text{Tr}(A^T A')}{\|A\|_F^2 + \|A'\|_F^2} \tag{6}$$

Bilinear autoencoders with comparable sparsity are near-identical (98% in `vanilla` and `ordered` Figure 8). For instance, a `vanilla` autoencoder attains over 99% similarity when varying seeds, demonstrating the consistency of the learning process. However, this similarity drops significantly

when comparing the densest and sparsest instances. This shows that sparsity penalties not only impact the latents but also the behaviour of the whole.

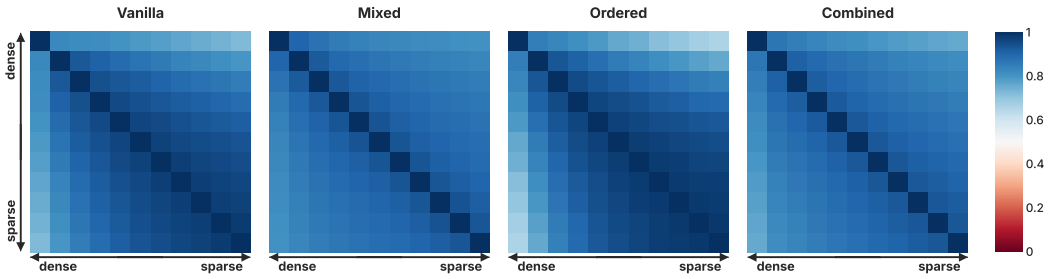

Figure 8: Frobenius similarity between bilinear autoencoders with density penalties for $\alpha \in [0, 1]$ (from Figure 6). This shows they learn comparable reconstructions across hyperparameters.

The Frobenius similarity above is global and basis-independent: it compares the exact reconstruction between autoencoders [2]. This is often not sufficiently precise since we may be interested in latent consistency: how similarly each latent is represented. Linearity is helpful again since it is possible to construct a matrix that maps how latents are represented on another autoencoder's basis. We explore this idea further and provide empirical results in Appendix H.

5 Conclusion

Summary. This paper uses bilinear autoencoders to decompose representations into polynomial latents. We argue that polynomial latents are suitable primitives for interpretability, as they can be analysed without inputs and help reveal composite geometries. We propose several extensions to induce latents with additional structure: sparsity, importance ordering, and clustering. Lastly, we show that our theory translates to practice by enabling the automated discovery of manifolds.

Future work. This paper studies bilinear autoencoders and highlights their advantages across tasks. Much work remains to rigourise results and explore the whole state space of possible applications:

- **Performing causal analysis and steering.** Using the interpretable manifolds as a basis for (automated) causal interventions to verify and nonlinearly steer model behaviour.
- **Enabling multi-layer interpretability.** Linearly mapping between the latent bases of stacked autoencoders to trace how features are composed through a deep network.
- **Investigating composite manifolds.** Designing sparsity penalties that align interacting latents to represent composite concepts, moving beyond the sparsity-induced single-latent alignment explored in this work.
- **Incorporating geometric biases.** Integrating architectural priors, such as bias terms, to restrict latents to specific (known) geometries.
- **Scaling to practical applications.** Applying these autoencoders to larger models and latent spaces to discover new circuits, debug model failures or verify safety properties.

Implications. This work moves beyond (near-)linear decompositions and promotes polynomials as algebraic primitives, as their computation and complexity can be analysed in a closed form. This can facilitate quantifying their description length, which is often intractable or ill-defined for other nonlinear operations without making assumptions [2]. This approach can help alleviate reliance on interpretable bases in favour of studying composition and interactions [16].

[2] This similarity can be adapted to account for ordering by including $M^\top M$ into the equation. This can be interpreted as a cumulative similarity but is not explored in this paper.

References

[1] 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, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/methods.html.

[2] Kola Ayonrinde, Michael T. Pearce, and Lee Sharkey. Interpretability as compression: Reconsidering sae explanations of neural activations with mdl-saes, 2024. URL https://arxiv.org/abs/2410.11179.

[3] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[4] 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. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[5] Bart Bussmann, Patrick Leask, and Neel Nanda. Batchtopk sparse autoencoders, 2024. URL https://arxiv.org/abs/2412.06410.

[6] Bart Bussmann, Noa Nabeshima, Adam Karvonen, and Neel Nanda. Learning multi-level features with matryoshka sparse autoencoders, 2025. URL https://arxiv.org/abs/2503.17547.

[7] David Chanin, James Wilken-Smith, Tomáš Dulka, Hardik Bhatnagar, Satvik Golechha, and Joseph Bloom. A is for absorption: Studying feature splitting and absorption in sparse autoencoders, 2025. URL https://arxiv.org/abs/2409.14507.

[8] Bob Coecke. Compositionality as we see it, everywhere around us, 2021. URL https://arxiv.org/abs/2110.05327.

[9] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models, 2023. URL https://arxiv.org/abs/2309.08600.

[10] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

[11] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks, 2017. URL https://arxiv.org/abs/1612.08083.

[12] Thomas Dooms, Ward Gauderis, Geraint Wiggins, and Jose Antonio Oramas Mogrovejo. Compositionality Unlocks Deep Interpretable Models. In *Connecting Low-Rank Representations in AI: At the 39th Annual AAAI Conference on Artificial Intelligence*, November 24.

[13] 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, 2022. URL https://arxiv.org/abs/2209.10652.

[14] Joshua Engels, Eric J. Michaud, Isaac Liao, Wes Gurnee, and Max Tegmark. Not all language model features are one-dimensionally linear, 2025. URL https://arxiv.org/abs/2405.14860.

[15] 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, 2024. URL https://arxiv.org/abs/2406.04093.

[16] Ward Gauderis, Thomas Dooms, Michael T Pearce, Kola Ayonrinde, José Oramas, and Geraint A. Wiggins. Compositional interpretability with deep tensor models.

[17] Sai Sumedh R. Hindupur, Ekdeep Singh Lubana, Thomas Fel, and Demba Ba. Projecting Assumptions: The Duality Between Sparse Autoencoders and Concept Geometry, March 2025.

[18] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks, 2006. URL https://www.science.org/doi/abs/10.1126/science.1127647.

[19] Patrik O. Hoyer. Non-negative matrix factorization with sparseness constraints. *J. Mach. Learn. Res.*, 5:1457–1469, December 2004. ISSN 1532-4435.

[20] Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

[21] Quoc V Le, Alexandre Karpenko, Jiquan Ngiam, and Andrew Y Ng. ICA with Reconstruction Cost for Efficient Overcomplete Feature Learning.

[22] Alexander Modell, Patrick Rubin-Delanchy, and Nick Whiteley. The origins of representation manifolds in large language models, 2025. URL https://arxiv.org/abs/2505.18235.

[23] Maxime Méloux, Silviu Maniu, François Portet, and Maxime Peyrard. Everything, everywhere, all at once: Is mechanistic interpretability identifiable?, 2025. URL https://arxiv.org/abs/2502.20914.

[24] Chris Olah. Mechanistic interpretability, variables, and the importance of interpretable bases, jun 27 2022. URL https://transformer-circuits.pub/2022/mech-interp-essay/index.html. Transformer Circuits Thread informal note.

[25] Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models, 2024. URL https://arxiv.org/abs/2311.03658.

[26] Gonçalo Paulo and Nora Belrose. Sparse autoencoders trained on the same data learn different features, 2025. URL https://arxiv.org/abs/2501.16615.

[27] Michael T. Pearce, Thomas Dooms, Alice Rigg, Jose M. Oramas, and Lee Sharkey. Bilinear mlps enable weight-based mechanistic interpretability, 2025. URL https://arxiv.org/abs/2410.08417.

[28] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[29] Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders, 2024. URL https://arxiv.org/abs/2407.14435.

[30] Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

[31] Xiaoqing Sun, Alessandro Stolfo, Joshua Engels, Ben Wu, Senthooran Rajamanoharan, Mrinmaya Sachan, and Max Tegmark. Dense sae latents are features, not bugs, 2025. URL https://arxiv.org/abs/2506.15679.

[32] TensorNetwork Project. Tensor diagram notation. https://tensornetwork.org/diagrams/, 2025. [Online; accessed 10-Aug-2025].

[33] Sean Tull, Robin Lorenz, Stephen Clark, Ilyas Khan, and Bob Coecke. Towards Compositional Interpretability for XAI, June 2024.

[34] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

A Experimental setup

We use the same hyperparameters across all discussed autoencoders. Since we only performed a few ablations, this setup is very likely suboptimal.

Data. We performed all experiments on layer 18 of `Qwen3-0.6B-Base` [34], a small but competent model. This lets us sweep over reasonably sized autoencoders and fully train on a tight compute budget. Specifically, the `vanilla` autoencoder trains in about 10 minutes on an RTX 4080. We successfully replicated these experiments on other models of varying sizes and ages (Appendix E).

Optimiser. We use the Muon optimiser [20], which we have found to significantly outperform Adam. This depends strongly on the setup, but it's roughly between $2\times$ and $10\times$ faster convergence. We use a trapezoidal learning rate schedule, cooling down to 0 over the last half of training. Finally, we use a linear warmup for α to prioritise reconstruction over sparsity near the start.

Architecture. We have found orthogonal initialisation (Appendix C) to perform better than Xavier initialisation. Orthogonal initialisation favours sparser solutions at a minute reconstruction cost. We use an expansion factor of 16, a mixing expansion of 2 (subsection 3.3), and $\alpha = 0.1$ by default.

Objective. We divide inputs with their L_2 norm and use the sum of squares error (SSE) over the more common RMS and MSE combination. This is done for convenience, removing some clutter from equations. Since the autoencoder is linear, this normalisation does not affect the optimisation objective beyond ensuring equivalently-scaled inputs, improving stability.

Batch size	32×2	**Optimiser**	Muon	**Initialisation**	Orthogonal
Sequence length	256	**Momentum**	No	**Model dimension**	1024
Steps	2^{10}	**Learning rate**	0.01	**Latent dimension**	16×1024
Total tokens	$\approx 8\text{M}$	**Schedule**	trapezoid	**Mix dimension**	2×1024
Shuffling	No	α **warmup**	256 steps	**Default** α	0.1

Figure 9: Hyperparameters related to data, the optimiser, and the architecture, respectively.

B Efficient evaluation of the loss

In subsection 2.3, we claim that using the kernel trick to train bilinear autoencoders is not expensive. Here, we discuss this claim in more detail. We first demonstrate that computing $\mathbf{B}^\top\mathbf{B}$ is efficient and then show that it can be calculated with an efficient contraction order and splitting to avoid memory allocation.

In short, the kernel matrix of a bilinear autoencoder can be efficiently evaluated by recasting it as an element-wise multiplication of two matrices.

$$BB^\top = (LL^\top) \odot (RR^\top) \tag{7}$$

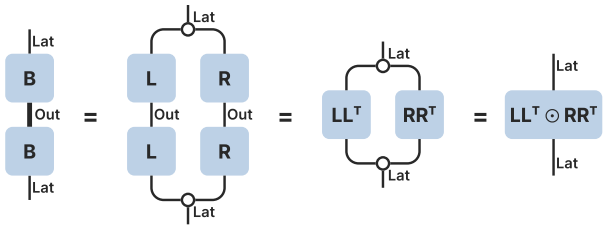

Figure 10: Diagrammatic equation on how to compute a bilinear autoencoder's kernel.

This perhaps also provides intuition on the ordering kernel matrix, which is also efficiently computable since it is simply an additional element-wise multiplication. The extra bottleneck in the `mixed` autoencoder only adds a matrix to the contraction, shrinking the kernel matrix.

While this trick reduces computational complexity, the kernel matrix is quadratic in the number of latents in memory, which can still be a bottleneck. However, it can be evaluated by splitting the Lat $\otimes$ Lat space into n blocks $s_1 \oplus s_2 \oplus \cdots \oplus s_n$, which avoids full materialisation.

$$\mathbf{f}^\top BB^\top \mathbf{f} = \sum_{s_i, s_j} \mathbf{f}_{s_i} \left(BB^\top\right)_{s_i s_j} \mathbf{f}_{s_j}$$

This can be additionally improved since only the symmetric part of $BB^\top$ contributes. Hence, we need only sample the lower or upper triangular part, speeding up computation by a factor 2.

This technique extends to all proposed autoencoder variants. In the `mixed` case, the down projection is one extra step in the kernel evaluation. In the `ordered` case, the mask matrix can be element-wise multiplied with each block.

C Autoencoder pseudo code

We provide distilled pseudo code for the initialisation and forward pass of the vanilla autoencoder. The intuition for why this works is provided in Appendix B. The full code can be found at `https://github.com/tdooms/bae`.

```
def __init__(self, config):
    self.left = nn.init.orthogonal(config.d_features, config.d_model)
    self.right = nn.init.orthogonal(config.d_features, config.d_model)

def forward(self, acts: Tensor['batch seq dims'], alpha: float):
    # Normalise the inputs and compute the features
    acts = acts * acts.square().sum(-1, keepdim=True).rsqrt()
    features = linear(acts, self.left) * linear(acts, self.right)

    # Compute the density and regularisation term
    density = hoyer(features, dim=(0, 1)).mean()

    # Compute constituent parts of the reconstruction error
    kernel = (self.left @ self.left.T) * (self.right @ self.right.T)
    recons = einsum(f, f, kernel, "... h1, ... h2, h1 h2 -> ...")
    cross = f.square().sum(-1)

    # Compute the error and loss
    error = (recons - 2 * cross + 1.0).mean()
    loss = error + alpha * density
    return loss, dict(error=error, density=density)
```

D Comparison of reconstruction error

We compare the reconstructions of all bilinear autoencoders across layers with a standard TopK ($k = 50$) autoencoder [15] baseline trained using the same setup (Figure 11). This baseline comparison is

unfair since the bilinear variants reconstruct the product input space rather than the inputs themselves. We work out the discrepancy, starting from the TopK reconstruction error:

$$s \;=\; \|\mathbf{x} - \hat{\mathbf{x}}\|^2 \;=\; \|\mathbf{x}\|^2 - 2\mathbf{x}^\top \hat{\mathbf{x}} + \|\hat{\mathbf{x}}\|^2.$$

Due to the input normalisation (Appendix A), we have $\|x\|^2 = 1$ and $\|\hat{x}\| \approx 1$, so we can approximate the quadratic reconstruction S given the regular reconstruction as follows.

$$\begin{aligned} S &= \|\mathbf{x} \otimes \mathbf{x} - \hat{\mathbf{x}} \otimes \hat{\mathbf{x}}\|^2 \\ &= 1 - 2(\mathbf{x}^\top \hat{\mathbf{x}})^2 + \|\hat{\mathbf{x}}\|^4 \\ &= \tfrac{1}{2}(1 - \|\hat{\mathbf{x}}\|^2)^2 + s(1 + \|\hat{\mathbf{x}}\|^2) - \tfrac{1}{2}s^2 \\ &\approx 2s - \tfrac{1}{2}s^2 \end{aligned} \tag{8}$$

For fair comparison, the baseline error should roughly be doubled ($s^2 \approx 0$ when small). Consequently, Figure 11 shows that bilinear autoencoders achieve better reconstruction than the baseline across most model layers and variants in the product space. This suggests that bilinear autoencoders can leverage decodable structure in interactions between input elements to improve reconstruction.

Another interesting observation is that `mixed` autoencoders outperform their `vanilla` counterparts everywhere except the first model layers. The `mixed` autoencoder adds a bottleneck, indicating that latents are better described as composites than independently. The fact that this doesn't happen in the first few layers may indicate these composite structures have not yet formed, but this is purely speculation.

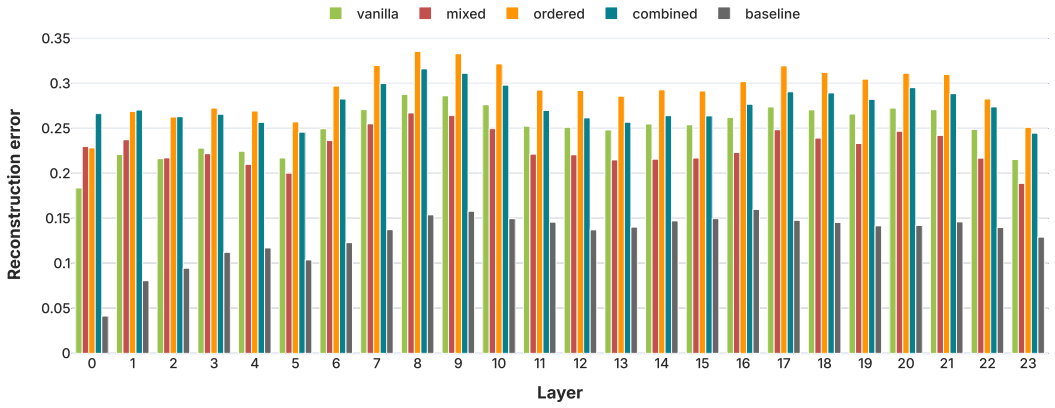

Figure 11: Reconstruction error across layers and autoencoder variants. The baseline comparison is unfair and should be doubled (Equation 8). Bilinear autoencoders often outperform the baseline in product space reconstruction.

E Anecdata and miscellaneous ablations

Convergence. All autoencoders are trained on 16M tokens, which takes a few minutes. This is sufficient to get highly interpretable features and interesting manifolds. While we lack the computational resources to train on billions of tokens, we train a $32\times$ `mixed` autoencoder on 256M tokens. Training metrics decrease steadily throughout training, down to an SSE of 0.14. This makes us optimistic that bilinear autoencoders can scale properly with compute.

Ordering. When the learning rate decreases, ordered autoencoders exhibit interesting learning dynamics near the end of training. The reconstruction error increases, sparsity remains the same, and loss decreases. This likely indicates that its features are being reshuffled and perhaps hints that the training process struggles with this task under certain circumstances and may require cyclic learning rates to alternate shuffling and learning.

Models. `Qwen3` models use the modern SwiGLU [34], which aligns well with bilinear encoders. We verify whether bilinear autoencoders work well on older models like GPT2-small [28] and Pythia-160M [3] and, more importantly, whether they produce interpretable features. The answer is positive even though the reconstruction error is about 20% higher on equivalent setups. More broadly, we find that more competitive models of similar size yield lower reconstruction errors.

Gates. Despite removing many appealing properties, we experimented with using nonlinearities in encoders to reflect an actual GLU. While this (slightly) reduces sparsity, it also yielded worse reconstructions, resulting in overall higher loss. Furthermore, we also experimented with applying the activation function after the element-wise multiplication. Using TopK required setting $k > 500$ to get competitive reconstructions with other bilinear autoencoders. In both cases, we uncoupled the encoder and decoder since their roles are no longer identical.

Bounded. The imposed latent bottleneck by reconstructing the squared space is bounded by the rank of $\mathbf{x} \otimes \mathbf{x}$. In theory, increasing the latent space beyond the size of the product space cannot reduce the reconstruction error any further, unless additional assumptions such as sparsity are imposed [21]. This insight can be used to predict the best achievable reconstruction error given random inputs for a given size. We can use this as a baseline to discern whether scaling the latent amount simply reconstructs noise or still finds structure. We find autoencoders up to an expansion factor 64 outperformed this random baseline, but did not scale this experiment further.

Muon. All experiments use the Muon optimiser [20] by default. It's hard to overstate the improvement over Adam; it often reduces training time by an order of magnitude. Furthermore, it's much more tolerant to changes in hyperparameters and changes in training setup. This was not the case for Adam in our experience.

F Reconstruction with ordered latents

This section empirically examines the effect of imposing an ordering on latents. Our results demonstrate that ordering influences the final reconstruction (Figure 11), highlighting a trade-off that deserves closer analysis. Here, we show that ordered autoencoders achieve lower reconstruction errors significantly faster than their baseline (Figure 12).

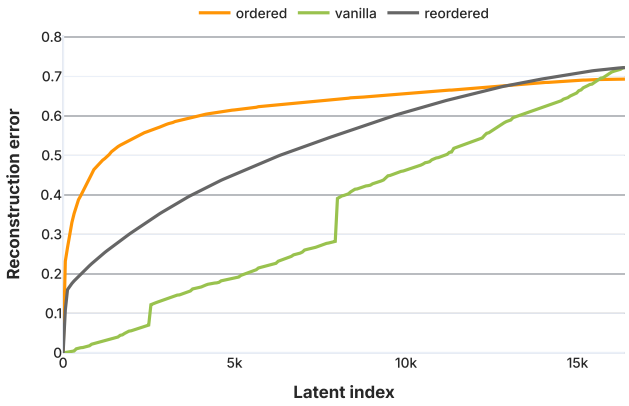

Figure 12: Reconstruction errors for all prefix subsets of latents on two autoencoder variants. The 'reordered' baseline uses the best greedy post-hoc ordering for the vanilla reconstruction for comparison. This is computed by recursively adding the latent which improves reconstruction the most.

The trained `ordered` and `combined` autoencoders use cumulative reconstruction errors. Hence, the first latent's reconstruction is often orders of magnitude more important than the last, which may be too 'strong'. Instead, one could consider a gentler schedule that only puts mild penalties on later latents. However, we have not yet experimented with how this impacts the latents' behaviour.

G Zooming in on a latent

This section discusses the first latent in the `vanilla` autoencoder without sparsity penalty (shown as the most dense model in Figure 6). Its activations correspond to possessive pronouns ("their", "her", "its"). Due to the lack of a sparsity-inducing activation function and symmetry about the origin, there is a question of how selectively it activates. We sample 128k activations and plot their magnitude, showing that 99% are smaller than $2 \cdot 10^{-5}$ (Figure 13). Qualitative analysis shows this is a representative distribution for other latents; sometimes the tail is denser or longer, but this rough shape is always present.

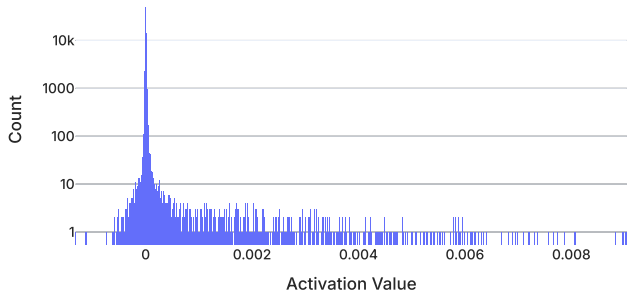

Figure 13: Histogram of 128k activations for the first latent of `vanilla` ($\alpha = 1$) autoencoder. Due to the logarithmic y-axis, the peak spans multiple orders of magnitude.

H Similarity between latent bases

subsection 4.3 showed that the learned structure is remarkably similar across autoencoders with varying hyperparameters. Yet, there is a question regarding local identifiability: do they learn similar latents across runs? This question can again be answered exactly for bilinear autoencoders.

This analysis is ordinarily based on the cosine similarity of decoder directions. Yet, this is incomplete as a latent's behaviour depends on its output direction (in the decoder) and its input direction (in the encoder). Furthermore, tiny changes in either encoder or decoder weights can strongly impact behaviour due to the activation functions, which are often ignored. Since bilinear latents can be studied independently of the inputs, their nonlinear behaviour can be analysed precisely.

A latent i is fully described by the matrix $B_i = \mathbf{l}_i \otimes \mathbf{r}_i$. Contracting two encoders like this $B^\top B'$ creates a (latent $\times$ latent) matrix which captures all pairwise similarities. When discussing identifiability, we seek this matrix to be as close to a permutation P as possible. In words, Equation 9 measures how much of the norm of $B^\top B'$ the best permutation P can recover.

$$\text{Similarity}_{\text{Perm}}(B, B') = \frac{\|B' P B^\top\|_F}{\|B^\top B'\|_F} \quad \text{where} \quad P = \underset{\text{permutation } P}{\arg\max}\ \text{Tr}(B' P B^\top) \tag{9}$$

Figure 14 show the resulting similarities for the `vanilla` autoencoder with varying sparsities.

The diagonal describes the self-similarity of an autoencoder and is not guaranteed to be 1 since decoders are the encoder's transpose (not inverse). It indicates the amount of interference (nonorthogonality) between latents and can be interpreted as how similar latents are after a 'round-trip' through an autoencoder. For the `vanilla` autoencoders from Figure 6, this value lies between 0.8 for densely activating autoencoders and 0.65 for their sparse counterparts.

The off-diagonal entries contain cross-similarities of two autoencoders. Note that this similarity is rather strict since it measures the similarity of the latents' bilinear forms, which is more sensitive than the cosine similarity of the vectors alone. This value is relatively constant and lies between 0.5 and 0.6.

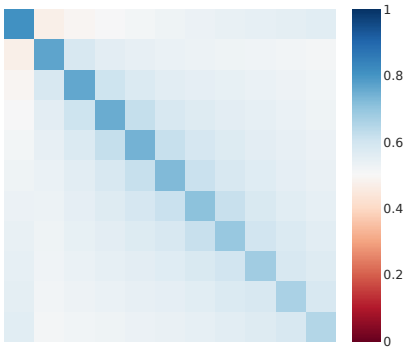

Figure 14: Permutation similarities (Equation 9) across `vanilla` autoencoder with varying sparsity penalties $\alpha \in [0, 1]$.

Lastly, we believe that consistency of the whole (Figure 8) matters more than latent identifiability (Figure 14) for bilinear autoencoders. In linear models, we rarely insist on aligning bases because our tools are not basis-dependent. Unlike nonlinear autoencoders, which fix their privileged latent basis using activation functions, bilinear autoencoders behave linearly in the product space, enabling rotation and combination of latents without leaving the bilinear form (subsection 2.2). Hence, analyses remain equivalent regardless of the used basis.