THE FEATURE-SPACE ALIGNMENT HYPOTHESIS FOR NEURAL NETWORK SPARSITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Why does something as simple as magnitude pruning retain accuracy even when most weights are removed, and why does performance eventually collapse at higher sparsities? This paper shows that the answer may lie in the ability to keep intact the representation of the original dense weight matrix when viewed in feature space, despite losing many weights to pruning. We study this question in a controlled setting where a linear embedding (representing prior layers) mixes input features across coordinates before classification, enabling direct analysis of the network's effective feature-space weights. We find that accuracy collapse under standard pruning coincides with divergence of the feature-space representation from its pre-pruning values. We then construct an optimization oracle that, given access to the embedding, selects a new sparse weight matrix, independent of the original, by directly preserving the effective feature-space matrix induced by the dense model. We show that under the same retraining budget, the oracle recovers performance at sparsity levels where standard pruning techniques degrade sharply, with consistent behavior across synthetic logic and more realistic tasks. Furthermore, we show that one can achieve oracle-level pruning performance even when the true embedding is not known by using a sparse autoencoder to recover its pruning-relevant structure, providing a path toward more general feature-aware sparsification. These results suggest a mechanistic explanation of sparsity and a guiding framework for feature-aware improvements in pruning methods.

1 INTRODUCTION

Weight sparsity has long been a goal in neural network research (LeCun et al., 1989; Hassibi & Stork, 1992; Hagiwara, 1994; Han et al., 2015; Frankle & Carbin, 2018; Frantar & Alistarh, 2023; Sun et al., 2023; Kong et al., 2026) and is increasingly relevant as models reach the trillion-parameter scale (Dubey et al., 2024; Team et al., 2025; Liu et al., 2025). Sparse models can reduce memory and inference cost, but it remains unclear what sparsification preserves at the level of learned features and computation. Most contemporary pruning methods nevertheless operate in parameter space, assigning importance to individual weights using magnitude-based scores or local loss sensitivity approximations and removing parameters that appear least important (Hassibi & Stork, 1992; Sun et al., 2023; Frantar & Alistarh, 2023). This implicitly assumes that the relevant underlying mechanisms, including features, circuits, and computations, are well-aligned with individual parameters.

However, growing evidence suggests that learned features can be entangled (Goh et al., 2021; Olah et al., 2020; Arora et al., 2018; Mu & Andreas, 2020; Jermyn et al., 2022; Gurnee et al., 2024; Dreyer et al., 2024), with multiple features sharing parameters and individual parameters contributing to many features (Elhage et al., 2022; Adler & Shavit, 2024). Here, weight-based heuristics may be misaligned with feature preservation. Thus, pruning can remove or distort computation even when the least important parameters are removed by standard criteria. Indeed, entangled neurons can be highly sensitive to pruning despite having small weights (Sawmya et al., 2025), suggesting a mismatch between parameter-level criteria and feature-level importance. This misalignment may contribute to brittleness when pushing to higher sparsity, where current algorithms often cause sharp degradation (Agarwalla et al., 2024) and recovery can require substantial fine-tuning (Lee et al., 2025). We hypothesize that the sparsity barrier reflects this mismatch between parameter-level pruning objectives and feature-level structure, which we term the *feature-space alignment hypothesis*.

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

To isolate this misalignment, we construct a controlled setup that contrasts physical-weight sparsity with preservation of the effective feature-space mapping, exposing latent structure hidden from raw parameters alone. Our goal is not an immediately deployable pruning method, but a mechanistic account of what sparsification preserves or disrupts, where weight-space heuristics diverge from basis-aware selection, and a framework for using such information to better guide pruning. On a Boolean task with known composition, pruning succeeds when it preserves the organization of learned latent features, while performance collapse coincides with degradation of this task-aligned representation. Using an optimization oracle to instead select sparse weight matrices that preserve this mapping recovers performance where standard methods fail, with similar advantages across more realistic tasks. Furthermore, sparse topology constrains recoverable performance: feature-aligned masks may underperform with inherited dense weights yet yield stronger recovery after fine-tuning. Compensatory updates to the surviving weights provide complementary one-shot improvements. Finally, we train sparse autoencoders (SAEs) on observed activations and use their decoder dictionaries as estimated feature bases. SAE-based pruning closely tracks performance obtained with oracle access to the true embedding, suggesting that direct access to the underlying feature basis is not required.

2 Related Work

Despite extensive work on both pruning and mechanistic interpretability, the two have largely progressed separately: pruning typically acts in parameter space and evaluates success through efficiency and end-task performance, while interpretability studies internal mappings expressed in distributed feature bases not aligned with individual weights or neurons. We connect these perspectives by arguing that this misalignment can be a primary bottleneck for sparse performance, making it central to understanding when sparsification succeeds or fails.

Sparsity Pruning typically identifies unimportant parameters via heuristics or local sensitivity measures. Magnitude pruning (Hagiwara, 1994) and second-order methods such as Optimal Brain Damage and Optimal Brain Surgeon (LeCun et al., 1989; Hassibi & Stork, 1992) show that many weights can be removed with limited immediate loss. Combinatorial approaches have been explored, but prior work optimizes OBS-style loss objectives over weights rather than sparsifying with respect to feature-space representations (Yu et al., 2022). At larger scales, one-shot methods demonstrate substantial practical sparsity (Frantar & Alistarh, 2023; Sun et al., 2023), yet pruning outcomes can be brittle and sensitive to training details (Gale et al., 2019; Liu et al., 2018). We provide a mechanistic account of this brittleness: under learned changes of basis, parameter-space saliency can be systematically misaligned with the feature-space directions implementing the computation.

Interpretability Mechanistic interpretability argues that networks can represent more features than neurons by storing them in a distributed or superposed form (Elhage et al., 2022; Adler & Shavit, 2024). Dictionary learning and sparse autoencoder methods recover sparse feature directions from dense activations, revealing feature vocabularies far larger than the neuron basis (Bricken et al., 2023; Templeton, 2024; Huben et al., 2023) and echoing classical sparse coding, where sparsity is defined relative to a learned dictionary (Olshausen & Field, 2004). However, these tools are mainly used to measure feature extractability (Alain & Bengio, 2016), not to guide compression. Only recently has work examined the interaction between distributed representations and sparsity in weights (Sawmya et al., 2025) and activations (Kong et al., 2025). We build on this by arguing that sparsity should be understood in the feature-aligned space of computation by showing how misalignment between feature- and parameter-space induces apparent sparsification limits.

3 Methodology

Model architecture We model a single layer of a network using a canonical setup, with an added embedding layer as established in Adler et al. (2025). Our model consists of a linear projection layer which feeds into a single hidden layer classifier. Formally, given inputs $\mathbf{x} \in \mathbb{R}^{d_{in}}$, we first apply a learnable linear projection $\mathbf{C}_0 \in \mathbb{R}^{d \times d_{in}}$ to obtain a transformed representation $\mathbf{x}' = \mathbf{C}_0\mathbf{x}$. This initial transformation simulates the role of prior layers in a deep network, especially for rotations.

Following this projection, the network computes output logits $\mathbf{z}$ via $\mathbf{z} = \mathbf{W}_2 \cdot \mathrm{ReLU}(\mathbf{W}_1\mathbf{x}' + \mathbf{b}_1) + \mathbf{b}_2$, where $\mathbf{W}_1 \in \mathbb{R}^{h \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{C \times h}$ are the weight matrices for the hidden and output layers,

respectively, and $\mathbf{b}_1, \mathbf{b}_2$ are biases. All layers including $\mathbf{C}_0$ are initialized with random weights, and we jointly optimize all parameters using binary cross-entropy loss in the case of the DNF boolean formulas or categorical cross-entropy loss for the more realistic tasks. To study the effect of feature-weight alignment in isolation, we intentionally avoid introducing dimensionality expansion or compression. For all experiments, we use a square linear embedding $\mathbf{C}_0$ whose output dimensionality matches its input dimensionality (the number of literals in the case of the boolean formula), and set the width of $\mathbf{W}_1$ to the same value. Full training and finetuning details in section A.2.

Analytical formulation The inclusion of $\mathbf{C}_0$ is central to our experimental design. Following the factorization proposed by Adler et al. (2025), we view training as approximately learning $\mathbf{W}_1 \approx \mathbf{C}_1\mathbf{C}_0^{-1}$, where $\mathbf{C}_1$ represents the feature-space computation. In this view, $\mathbf{W}_1$ first disentangles the relevant features from the dense mixture $\mathbf{x}'$ induced by $\mathbf{C}_0$ and then performs the computation in feature space. This setup allows us to isolate how pruning the physical weights $\mathbf{W}_1$ affects the underlying feature-space computation. Thus, $\mathbf{C}_0$ serves two roles: it mixes the input features into the representation seen by $\mathbf{W}_1$ and, because the trained matrix is explicitly available, provides oracle access to the effective mapping $\mathbf{C}_1 = \mathbf{W}_1\mathbf{C}_0 \in \mathbb{R}^{h \times d_{in}}$. We refer to the basis induced by $\mathbf{C}_0$ as the true feature basis in our controlled setting. Preserving $\mathbf{C}_1$ preserves the layer's effective mapping; we therefore use $\mathbf{C}_1$ as our primary object of analysis, allowing us to track the emergence and preservation of feature structure while distinguishing it from sparsity in the physical weight space $\mathbf{W}_1$ (fig. 3). In section 4.4, we show that a dictionary learned only from observed activations serves as an effective substitute for oracle access to this basis, recovering most of its pruning benefit.

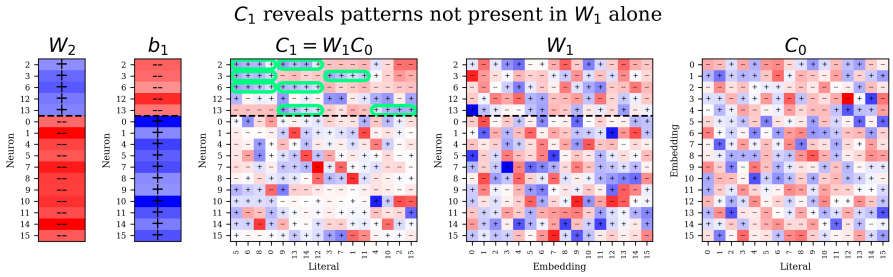

Figure 1: Visualizing weights in $\mathbf{C}_1$ reveals codes (section 4.1) that are not present in $\mathbf{W}_1$. When the weights in $\mathbf{C}_1$ are organized by the literals within the Boolean formula upon which the model is trained, clear codes emerge in $\mathbf{C}_1$ that reveal the inner computations in the feature space of the model. Sets of weights circled in green indicate 4P-positive codes, as explained in section 4.1.

To make this intuition concrete, consider the example shown in fig. 1. The figure displays the parameters of a fully dense model ($\mathbf{C}_0$, $\mathbf{W}_1$, $\mathbf{B}_1$, and $\mathbf{W}_2$) after 25 epochs of training (section A.2), along with the induced feature-space matrix $\mathbf{C}_1 = \mathbf{W}_1\mathbf{C}_0$. The model was trained, as we will elaborate later, to learn a DNF formula of 4-variable clauses. For visualization, both $\mathbf{W}_1$ and $\mathbf{C}_1$ are ordered by ground-truth clauses along the horizontal axis, and by neurons with positive output weights in $\mathbf{W}_2$ along the vertical axis. We can see in green highlight in $\mathbf{C}_1$ the codes that emerge in rows with positive $\mathbf{W}_2$ neurons: 4 consecutive positive weights in a row with a corresponding negative bias. This is a soft Boolean *AND* of the specific clause $x_i \wedge x_{i+1} \wedge x_{i+2} \wedge x_{i+3}$, computed via $\mathrm{ReLU}(w_i x_i + w_{i+1}x_{i+1} + w_{i+2}x_{i+2} + w_{i+3}x_{i+3} - b)$. Notably, the codes emerge in feature space, and cannot be seen in the model's original weight space where magnitude pruning will happen.

As we will show, sparsity in $\mathbf{W}_1$ preserves behavior when a sparse matrix $\mathbf{W}_1'$ satisfies $\mathbf{W}_1'\mathbf{C}_0 \approx \mathbf{C}_1$, that is, while $\mathbf{W}_1$ changes, the coding in $\mathbf{C}_1$ remains intact. Exact equality is generally unattainable, and pruning induces a trade-off that manifests as reconstruction error in feature-space. Standard pruning preserves $\mathbf{C}_1$ only when its parameter-space criteria align with this objective. This motivates our *feature-space alignment hypothesis*: successful sparsification may require identifying a sparse weight-space realization that preserves the original feature-space computation.

DNF example and code formulation To mechanistically investigate our claims in a quantifiable way, we begin with a controlled setting where the target features are known exactly and can be tracked throughout training and sparsification (section 4.1). We note that our framework does not

need to know the features; this is purely for clarity of analysis. Concretely, we study learning k-uniform read-once monotone DNF formulas (O'Donnell, 2014), a standard benchmark for compositional structure (Valiant, 1984; Daniely et al., 2016; Bronstein et al., 2022). The target function is $f(x) = \bigvee_{j=1}^{m/4} \bigwedge_{\ell \in S_j} x_\ell$, where $\{S_j\}$ form a partition of $\{1, \ldots, m\}$ with $|S_j| = 4$ for all j.

Formally, for a given hidden neuron i and a target clause S_j, a code is defined as the joint sign pattern of the feature-space weights $\{(\mathbf{C}_1)_{i,\ell} \mid \ell \in S_j\}$. This combined pattern determines both the literals to which the neuron responds and whether its activation contributes positively or negatively to the output logit. We focus specifically on the "4P-positive" codes computing a soft Boolean *AND*: a neuron with positive effective weights for all four literals of a clause (4P), a negative bias ($(\mathbf{b}_1)_i < 0$) and a positive output weight ($(\mathbf{W}_2)_i > 0$). We also evaluate on Scikit-learn Digits (Pedregosa et al., 2011) and four UCI benchmarks: Letter Recognition (Slate, 1991), Pendigits (Alpaydin & Alimoglu, 1996), Satimage (Srinivasan, 1993), and Waveform (Breiman & Stone, 1988).

4 Results

4.1 Performance degradation at increasing sparsities tracks feature loss

We investigate the structural properties of the learned solution within the effective weight matrix $\mathbf{C}_1$. In this section, we track the formation of clause-detecting codes during training and evaluate their response to pruning. We study this behavior across problem instances with 16, 32, 48, and 64 literals to assess how feature formation and pruning effects scale with task complexity.

As training progresses and training loss decreases, we observe the consistent emergence of the dominant 4P-positive code (fig. 11), in line with prior work (Adler et al., 2025). To explain these codes more in depth, functionally, this configuration implements a soft logical *AND* (Gupta, 1999): the neuron remains inactive unless all four literals are present with sufficient strength to overcome the negative bias, at which point it activates and contributes positively to the output. As a result, the neuron selectively fires on inputs satisfying the clause, acting as a continuous relaxation of a Boolean 4-*AND* gate. While there are other important codes that handle near misses, we focus our subsequent analysis on the formation and preservation of 4P-positive codes.

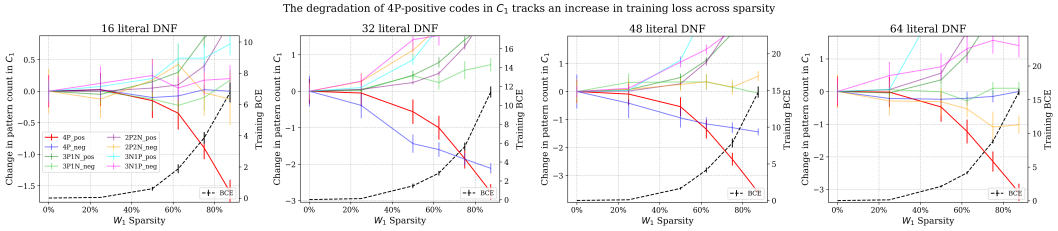

Figure 2: Code and performance degradation under sparsity. As $\mathbf{W}_1$ sparsity via magnitude pruning increases, an increase in loss on the training dataset tracks the loss of feature codes in $\mathbf{C}_1$. This supports our framework as one in which the performance of the network as a whole is reflected by computational codes that remain interpretable. 4N codes not shown since their effect after ReLU is zero. Error bars indicate one standard error of the mean. Results collected over ten trials.

We now use this framework to analyze the mechanism underlying performance degradation under sparsity. Specifically, we track the prevalence of 4P-positive codes in $\mathbf{C}_1$ as $\mathbf{W}_1$ is progressively sparsified via magnitude pruning (Hagiwara, 1994), which removes weights with the smallest absolute value. We keep the same sparsity per neuron. As sparsity increases, we observe a sharp increase in loss on the training dataset as well as a degradation of 4P-positive codes. Notably, this loss increase is not gradual: it closely mirrors the collapse of 4P-positive codes and aligns far more strongly with their degradation than with that of any other coding pattern, many of which increase in number (fig. 2). This correspondence suggests that 4P-positive codes are not incidental byproducts of training, but instead constitute the primary feature-level structures supporting computation.

To build intuition, fig. 3 visualizes the effective feature-space weights $\mathbf{C}_1$ at increasing sparsity levels in the physical weights $\mathbf{W}_1$. As expected, $\mathbf{C}_1$ remains numerically dense even under extreme

sparsification of $\mathbf{W}_1$ since it is obtained by composing a sparse matrix with the dense embedding $\mathbf{C}_0$, However, this density masks a substantial loss of structure: although many entries of $\mathbf{C}_1$ remain nonzero, the coherent, sign-consistent patterns corresponding to clause-detecting codes progressively degrade. Thus, the damage from sparsification does not primarily act by making $\mathbf{C}_1$ sparse, but by disrupting its alignment with the dense solution.

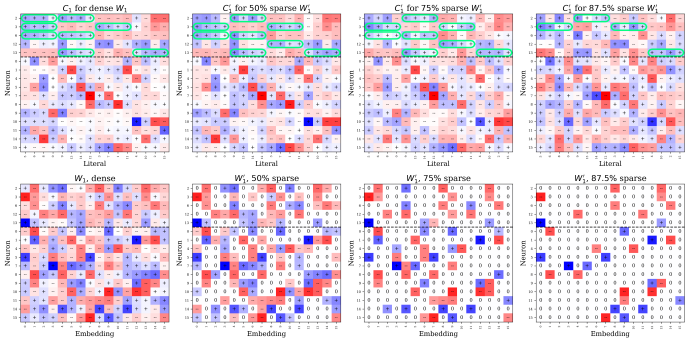

Figure 3: Visualization of $\mathbf{C}_1$ and $\mathbf{W}_1$ under sparsity. Even at high physical weight $\mathbf{W}_1'$ sparsity, $\mathbf{C}_1'$ remains fairly dense, as sparsity in parameter space does not translate to sparsity in feature space. This highlights that preserving the structure of $\mathbf{C}_1'$ is relevant. Dense model is same as fig. 1.

We repeat the same sparsification setup for the Scikit-learn Digits task. This dataset consists of 8×8 grayscale images of handwritten digits, providing a simple but non-synthetic benchmark that captures real-world visual structure while remaining computationally lightweight. Similarly to the DNF task, increasing sparsity rapidly destroys visible structure in $\mathbf{W}_1$, yet the induced feature-space matrix $\mathbf{C}_1$ remains reasonably close to the dense solution up to moderate sparsity levels, before degrading sharply at higher sparsities (fig. 4). This correlates with a drop in accuracy (fig. 17). This behavior may help explain why magnitude pruning works well at moderate levels. It also motivates explicitly preserving $\mathbf{C}_1$ rather than relying on implicit alignment of subsets of the original weights.

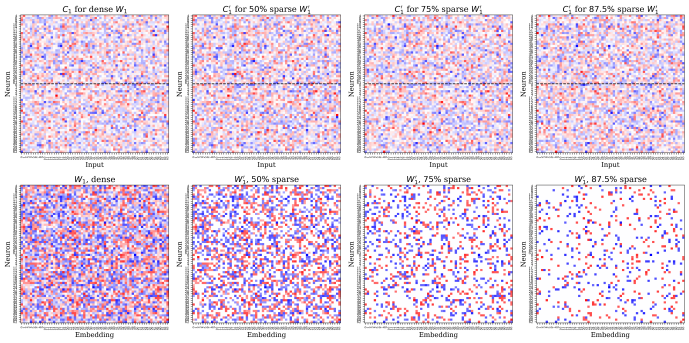

Figure 4: The effect of magnitude pruning on weight and feature-space representations, this time on Scikit-learn Digits. The scaled L_2 Frobenius norms between the dense and sparse $\mathbf{C}_1$ matrices is 0.38, 0.65, and 0.81 for 50%, 75%, and 87.5% $\mathbf{W}_1$ sparsity, respectively.

4.2 Directly optimizing for effective fidelity improves sparse performance

Having identified $\mathbf{C}_1$ as the relevant computational representation, we ask whether sparse performance improves by preserving the effective feature-space mapping rather than parameter-space properties. This preserves $\mathbf{C}_1$ as a whole and does not require identifying specific feature codes. We first use the trained $\mathbf{C}_0$ as an oracle to quantify the benefits of exact embedding information; the same preservation can be realized with an estimated dictionary, as we explore in section 4.4.

Minimizing divergence from $\mathbf{C}_1$ To characterize an empirical upper bound of sparse performance under this objective, we formulate an optimization problem that selects a sparse topology by

minimizing distortion in the feature-space. We pose this as a constrained reconstruction problem: given the effective weights $\mathbf{C}_1$ of a trained dense model, we search for a sparse physical weight matrix $\mathbf{W}_1'$ whose composition with $\mathbf{C}_0$ best approximates the effective weights in Frobenius norm.

Formally, we solve the mixed integer problem (MIP):

$$\begin{aligned} &\min_{\bar{\mathbf{W}}_1,\,\mathbf{M}} \quad \|\mathbf{W}_1\mathbf{C}_0 - \mathbf{W}_1'\mathbf{C}_0\|_F^2\,, \quad \text{where} \;\; \mathbf{W}_1' := \mathbf{M} \odot \bar{\mathbf{W}}_1 \\ &\text{subject to} \quad \sum_{j=1}^{d} \mathbf{M}_{ij} \le \tau, \;\; \forall i \in \{1,\dots,h\}; \;\; \mathbf{M}_{ij} \in \{0,1\}; \;\; \bar{\mathbf{W}}_1 \in \mathbb{R}^{h\times d} \end{aligned} \tag{1}$$

where $\mathbf{M}$ is a binary mask, $\bar{\mathbf{W}}_1$ contains the continuous weight variables, and τ controls per-neuron sparsity. We solve the optimization independently for each neuron for tractability and implement it in Python using CVXPY (Diamond & Boyd, 2016) with the SCIP (Hojny et al., 2025) solver.

Solving this objective yields a sparse approximation that is optimal in preserving the effective mapping (fig. 5). Notably, this formulation separates topology from weight values: the binary mask $\mathbf{M}$ identifies which physical connections are necessary to preserve feature-space alignment, while the continuous variables $\bar{\mathbf{W}}_1$ allow the surviving weights to adjust to compensate for pruned connections. As sparsity increases, the MIP solution departs further from the dense model in parameter space than magnitude pruning, while remaining closer in feature space. That is, the relative change in $\mathbf{W}_1$ is larger under MIP, but the change in $\mathbf{C}_1$ is smaller (fig. 5). Comparing the sparse topologies identified by this computational oracle with standard pruning methods isolates the gap due to topological misalignment: parameter-space heuristics can discard low-saliency weights that are nevertheless critical for preserving feature-level structure.

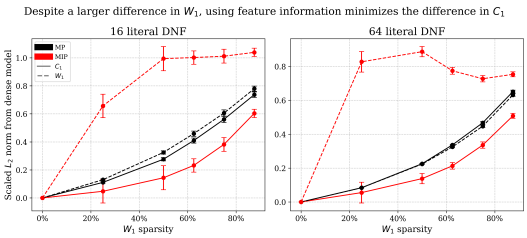

Figure 5: Divergence of $\mathbf{W}_1$ and $\mathbf{C}_1$. Despite MIP inducing a greater relative divergence in $\mathbf{W}_1$ from the dense model compared to MP, it induces a lesser divergence in $\mathbf{C}_1$. Error bars indicate one standard error of the mean. Results collected over ten trials. More configurations shown in fig. 12.

Effectiveness of maintaining features We compare the performance of the topology identified by our optimization oracle against three standard baselines: magnitude pruning (MP) (Han et al., 2015), Optimal Brain Damage (OBD) (LeCun et al., 1989), and Optimal Brain Surgeon (OBS) (Hassibi & Stork, 1992). The same sparsity level per neuron in $\mathbf{W}_1$ was enforced for all pruning techniques. MP removes weights in $\mathbf{W}_1$ with the smallest absolute values. OBD refines MP by incorporating a diagonal approximation to the Hessian, pruning weights estimated to cause the smallest increase in loss under a second-order approximation. OBS further extends this approach by accounting for interactions between weights via an approximate inverse Hessian, and applies a one-shot update to the remaining weights after pruning. At the scale studied here, OBS provides a particularly stringent second-order baseline, allowing us to apply loss-aware compensation directly rather than through scalability-driven approximations (Frantar & Alistarh, 2023). Full details provided in section A.3.

In contrast, **M**ixed **I**nteger **P**rogramming (MIP) directly optimizes for a sparse weight matrix $\mathbf{W}_1'$ by minimizing distortion in the effective feature-space mapping $\mathbf{W}_1'\mathbf{C}_0$ relative to the dense mapping $\mathbf{W}_1\mathbf{C}_0$, subject to a sparsity constraint (eq. (1)). Finally, the **F**eature **Lo**ss Hybrid (FLO) method uses the sparse mask identified by MIP while keeping the original weights fixed, followed by an OBS-style weight update restricted to this mask. This decouples topology selection from weight refinement, combining feature-aware sparsification with loss-based optimization. Each technique induces a distinct sparse mask on $\mathbf{W}_1$, reflecting different notions of parameter importance (fig. 13)

We first compare MP, OBD, OBS, and the proposed feature-aware MIP after pruning without any retraining. Across all problem sizes and sparsity levels, MIP and OBS achieve broadly comparable performance: both substantially outperform MP and OBD, but neither consistently dominates the other (fig. 6). This suggests that preserving feature-space structure and compensating for pruning-induced error via second-order updates address complementary aspects of sparsification.

Motivated by the strong performance of the OBS weight update, we introduce the FLO hybrid approach, which combines the MIP sparse topology with the OBS weight update rule. FLO consistently outperforms all other methods across sparsity levels, indicating that accurate identification of a feature-aligned mask and effective post-pruning weight compensation jointly contribute to improved sparse performance. This result further supports the view that sparsification failures can arise from both topological misalignment and suboptimal weight adjustment, and that addressing both may be necessary to approach dense performance. By explicitly targeting the reconstruction of $\mathbf{C}_1$, MIP identifies a sparse support that preserves the network's functional logic more effectively than saliency heuristics derived from the physical loss landscape. We repeat this experiment in deeper networks, where feature-space information continues to improve performance (section A.10).

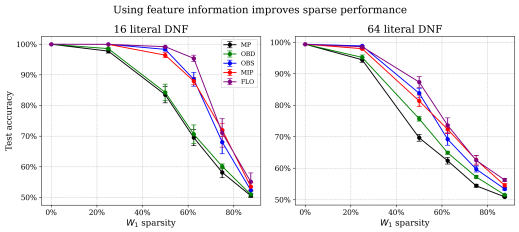

Figure 6: Feature-space information improves performance. Error bars indicate one standard error of the mean. Results collected over ten trials. More configurations shown in fig. 14.

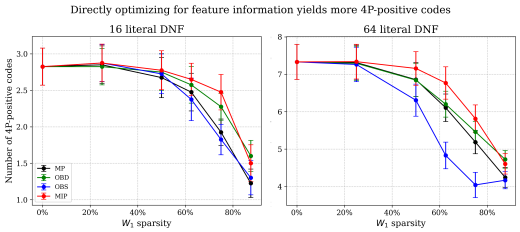

Figure 7: 4P-positive codes are better maintained by MIP. Error bars indicate one standard error of the mean. Results collected over ten trials. More configurations shown in fig. 15.

Additionally, we analyze the prevalence of 4P-positive codes under increasing sparsity. As shown in fig. 7, MIP consistently preserves a larger number of 4P-positive codes compared to MP, OBD, and OBS. Notably, this behavior emerges despite the fact that the MIP objective does not explicitly target these codes, but instead optimizes reconstruction of the full feature-space $\mathbf{C}_1$. The resulting retention of 4P-positive codes therefore arises as a byproduct of minimizing feature-space reconstruction error, rather than from direct supervision of any specific code pattern. We emphasize that this correlation is not causal. Rather, these codes serve as an interpretable proxy for the stability of functionally meaningful structure in $\mathbf{C}_1$. Their preferential preservation under MIP is indicative of a broader robustness of the learned feature organization, helping to explain why MIP-selected masks support higher recoverable performance after fine-tuning.

Mask topology is crucial To separate topological selection from weight optimization, we evaluate each method using only its sparse mask $\mathbf{M}$ and keep all surviving weights at their trained dense values (this only affects OBS and MIP). In this ablation, the MIP mask initially underperforms magnitude pruning, suggesting the oracle-optimized weight values are not robust to direct reuse. However, with the same fine-tuning budget and training to saturation, the MIP-derived topology consistently achieves the best final performance, especially at higher sparsities and for larger formulas (fig. 8).

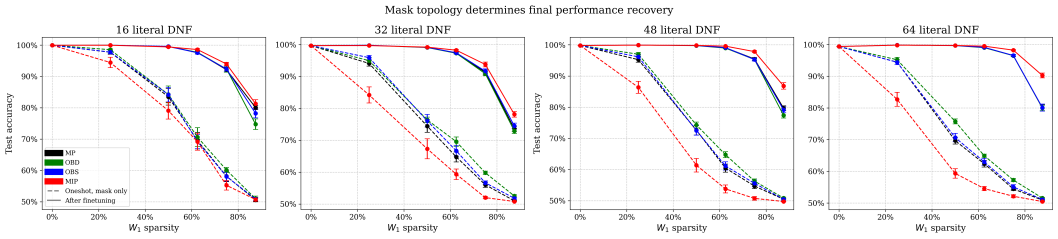

Figure 8: Mask topology determines performance recovery under sparsification. Applying just the sparse mask $\mathbf{M}$ to $\mathbf{W}_1$ without any weight adjustments yields subpar performance for MIP. However, with equal fine-tuning budget, the MIP mask allows the model to recover more test accuracy. Error bars indicate one standard error of the mean. Results collected over ten trials.

This behavior reveals a critical distinction between initialization quality and representational capacity. While the dense weight values are fragile under sparsity, the topology identified by the oracle provides a superior substrate for restoring the feature-space representation during fine-tuning. In contrast, standard pruning heuristics greedily preserve weights that are currently deemed salient, but often discard low-saliency connections that are nevertheless necessary to reconstitute the aligned feature basis. As a result, their fine-tuned performance saturates at a lower level despite equal optimization effort. Notably, they converge to similar performance ceilings after fine-tuning, suggesting that these heuristics identify broadly similar topologies that impose a shared limitation on feature recovery. In contrast, the MIP mask preserves structurally necessary connections that enable continued restoration of the effective basis, allowing it to reach a higher performance plateau under the same optimization budget. These results suggest that fine-tuning primarily restores lost feature alignment when the underlying topology permits it, rather than discovering entirely new feature structures.

We further observe that fine-tuning leaves the overall code structure largely intact. Codes that are present immediately after pruning tend to persist throughout fine-tuning, while codes that are absent in the post-pruning effective mapping are rarely created during fine-tuning, similarly suggesting that optimization primarily rebalances and amplifies surviving structure rather than reliably reconstructing missing features. These observations support the view that final recoverable performance is largely determined by whether the pruning step preserves a sufficiently rich code substrate (fig. 19). Additionally, we find that the benefits of feature-aware selection persist under hardware-relevant structured sparsity constraints and on the Digits task, where FLO remains the strongest-performing method despite the absence of canonical feature codes (section A.8).

4.3 Experiments on real-world datasets

We extend our analysis beyond DNF and Digits to four additional multiclass benchmarks from the UCI Machine Learning Repository: Letter Recognition, Pendigits, Satimage, and Waveform. These datasets span pixel-derived letter statistics, pen-stroke coordinates, multispectral satellite measurements, and noisy mixtures of waveform basis functions. As in the DNF experiments, the embedding output dimension matches the input dimension, while here the hidden layer contains 16 neurons.

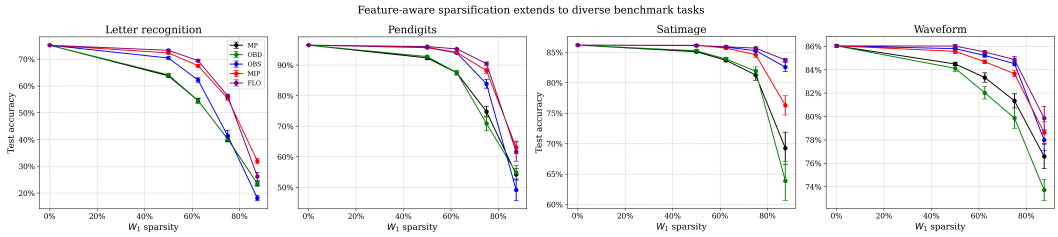

Figure 9: Improvements from feature-awareness extend to more realistic datasets. Error bars indicate one standard error of the mean. Results collected over ten trials.

Across all datasets and sparsity levels, either MIP or FLO achieves the highest mean accuracy (including ties), with generally greater preservation at higher sparsity levels as standard methods degrade more sharply. The strong performance of FLO further suggests that feature-aware topology selection and post-pruning weight adjustment are complementary. These results extend the feature-space alignment hypothesis beyond the synthetic DNF setting and its task-specific code structure.

4.4 Sparse autoencoders provide a pruning-relevant feature dictionary

The preceding experiments assume oracle access to the trained $\mathbf{C}_0$. In practice, only activations $\mathbf{x}'$ may be observable. Since an SAE decoder maps sparse latent features into activation space, Adler et al. (2025) suggest that it may serve as a surrogate for $\mathbf{C}_0$; we test whether its learned dictionary can substitute for $\mathbf{C}_0$ in the pruning objective. Specifically, we train a Top-K SAE following Gao et al. (2024) with latent width m and sparsity K, encoding $\mathbf{z} = \mathrm{ReLU}(\mathrm{TopK}(\mathbf{W}_{\mathrm{enc}}(\mathbf{x}' - \mathbf{b}_{\mathrm{dec}}) + \mathbf{b}_{\mathrm{enc}}, K))$ and reconstructing $\hat{\mathbf{x}}' = \mathbf{W}_{\mathrm{dec}}\mathbf{z} + \mathbf{b}_{\mathrm{dec}}$. Decoder columns are unit-normalized; training combines reconstruction error with AuxK loss to reduce dead latents. More details in section A.11.

Since $\mathbf{x}' \approx \mathbf{W}_{\text{dec}}\mathbf{z} + \mathbf{b}_{\text{dec}}$, $\mathbf{W}_{\text{dec}}$ provides an activation-derived dictionary and $\mathbf{W}_1\mathbf{W}_{\text{dec}}$ represents the next layer's computation on its latent features. We replace $\mathbf{C}_0$ in the oracle objective with $\tilde{\mathbf{C}}_0 = \mathbf{W}_{\text{dec}}$ and minimize $\|\mathbf{W}_1\mathbf{W}_{\text{dec}} - \mathbf{W}_1'\mathbf{W}_{\text{dec}}\|_F^2$ subject to the constraints in eq. (1). We call this method MIP-SAE and call its mask followed by OBS compensation FLO-SAE. Dictionary learning uses only samples of $\mathbf{x}'$, without access to $\mathbf{C}_0$ or the original inputs $\mathbf{x}$. We use the same representative configuration, $m = 32$ and $K = 6$, for every dataset and sparsity level rather than tuning it per condition. Full sensitivity sweeps reveal a trade-off between dictionary capacity and feature selectivity: successful pruning requires sufficient dictionary width and an intermediate K, as either overly restrictive or insufficiently sparse codes can degrade pruning performance (figs. 20 to 23).

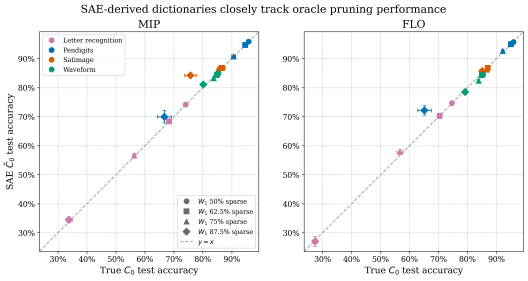

Figure 10: Test accuracy using SAE-derived versus true mapping across datasets and sparsity levels. Error bars indicate one standard error of the mean. Results collected over ten trials.

MIP-SAE and FLO-SAE closely track their counterparts using the true $\mathbf{C}_0$ (fig. 10). Across the 32 method–dataset–sparsity combinations, 26 differ by at most one percentage point, with a median absolute difference of 0.44 points and a mean absolute difference of 0.99 points. These results indicate that a dictionary learned only from observed activations can recover most of the benefit of feature-aware sparsification without privileged access to $\mathbf{C}_0$.

5 Conclusion and Limitations

In this work, we connect interpretability and sparsity via a simple failure mode for pruning: standard algorithms act in the physical weight basis $\mathbf{W}_1$, while the computation is naturally expressed in an effective, feature-aligned basis $\mathbf{C}_1 = \mathbf{W}_1\mathbf{C}_0$. When representations are distributed, these bases can become misaligned, so weight-space importance need not track functional importance; $\mathbf{C}_1$ is therefore the natural object for analyzing and preserving sparsity.

In controlled 4-DNF experiments, accuracy degrades alongside the loss of specific feature codes in $\mathbf{C}_1$. Across both the boolean setting and more realistic tasks, selecting sparse masks to explicitly preserve the effective mapping recovers high performance at sparsity levels where magnitude- and Hessian-based baselines degrade sharply. Ablations suggest topology selection is the main bottleneck: even masks with worse one-shot accuracy can fine-tune better if they retain a more viable substrate for features. Fine-tuning largely rebalances what survives, but does not reliably recreate missing feature codes. Thus, attainable sparse performance is set primarily by whether the initial pruning preserves the required code structure. These effects persist in deeper networks and under hardware-relevant structured sparsity. Crucially, replacing privileged access to $\mathbf{C}_0$ with an SAE dictionary learned only from observed activations recovers most of the oracle benefit, indicating that sparsification in a feature-aware manner does not require direct knowledge of the true feature basis.

Important limitations nevertheless remain. Although the SAE removes the requirement for privileged access to $\mathbf{C}_0$, both the oracle and SAE-derived methods still rely on mixed-integer optimization, which is NP-hard (Karp, 2009) and therefore unsuitable for pruning large models directly. The SAE approximation also depends on learning a sufficiently expressive and selective dictionary, and its reliability has so far been demonstrated only in our controlled experimental setting. Developing scalable approximations to the feature-preservation objective and testing activation-derived feature bases on modern transformers and downstream tasks are therefore natural directions for future work.

To close, we view the models studied here as scientific instruments. They serve as settings in which this specific mechanism can be made visible and isolated, even if they are not substitutes for more realistic architectures. We hope this work can therefore be read as establishing a mechanism where it is directly observable, while providing a conceptual framework–learning feature bases from activations and using them to guide and align sparsification–for larger-scale investigation.

AI USE STATEMENT

In this work, we used generative AI tools to support research ideation and execution, including providing feedback on aspects of the methodology and assisting with the implementation of portions of the experimental and data analysis code. We also used generative AI tools as writing aids during the drafting and revision process. We directed each iteration and revised, incorporated, or rejected AI-generated suggestions to improve the manuscript's clarity. We have not used generative AI tools to generate synthetic datasets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, assist with translation, clean and reformat datasets, or interpret results. We have reviewed all AI-assisted work: all AI-assisted manuscript content was reviewed and revised by the authors, and all AI-assisted experimental and data analysis code was reviewed, verified, and tested for correctness by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Micah Adler and Nir Shavit. On the complexity of neural computation in superposition. *arXiv preprint arXiv:2409.15318*, 2024.

Micah Adler, Dan Alistarh, and Nir Shavit. Towards combinatorial interpretability of neural computation. *arXiv preprint arXiv:2504.08842*, 2025.

Abhinav Agarwalla, Abhay Gupta, Alexandre Marques, Shubhra Pandit, Michael Goin, Eldar Kurtic, Kevin Leong, Tuan Nguyen, Mahmoud Salem, Dan Alistarh, et al. Enabling high-sparsity foundational llama models with efficient pretraining and deployment. *arXiv preprint arXiv:2405.03594*, 2024.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Ethem Alpaydin and Fevzi Alimoglu. Pen-based recognition of handwritten digits. *UCI Machine Learning Repository*, 10:C5MG6K, 1996.

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics*, 6:483–495, 2018.

L Breiman and C Stone. Waveform database generator (version 1). *UCI Machine Learning Repository*, 10:C5CS3C, 1988.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, et al. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2, 2023.

Ido Bronstein, Alon Brutzkus, and Amir Globerson. On the inductive bias of neural networks for learning read-once dnfs. In *Uncertainty in Artificial Intelligence*, pp. 255–265. PMLR, 2022.

Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. *Advances in neural information processing systems*, 29, 2016.

Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Maximilian Dreyer, Erblina Purelku, Johanna Vielhaben, Wojciech Samek, and Sebastian Lapuschkin. Pure: Turning polysemantic neurons into pure features by identifying relevant circuits. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8212–8217, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

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.

Gabriel Goh, Nick Cammarata, Chelsea Voss, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. *Distill*, 6(3):e30, 2021.

Madan M Gupta. *Soft computing and intelligent systems: theory and applications*. Elsevier, 1999.

Wes Gurnee, Theo Horsley, Zifan Carl Guo, Tara Rezaei Kheirkhah, Qinyi Sun, Will Hathaway, Neel Nanda, and Dimitris Bertsimas. Universal neurons in gpt2 language models. *arXiv preprint arXiv:2401.12181*, 2024.

Masafumi Hagiwara. A simple and effective method for removal of hidden units and weights. *Neurocomputing*, 6(2):207–218, 1994.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

Christopher Hojny, Mathieu Besançon, Ksenia Bestuzheva, Sander Borst, João Dionísio, Johannes Ehls, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Adrian Göß, et al. The scip optimization suite 10.0. *arXiv preprint arXiv:2511.18580*, 2025.

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

Adam S Jermyn, Nicholas Schiefer, and Evan Hubinger. Engineering monosemanticity in toy models. *arXiv preprint arXiv:2211.09169*, 2022.

Richard M Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pp. 219–241. Springer, 2009.

Linghao Kong, Angelina Ning, Micah Adler, and Nir Shavit. Negative pre-activations differentiate syntax. *arXiv preprint arXiv:2509.24198*, 2025.

Linghao Kong, Inimai Subramanian, Micah Adler, Dan Alistarh, Dan Gutfreund, and Nir Shavit. The sparsity whisperer. *arXiv preprint arXiv:2608.06630*, 2026.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Kwanhee Lee, Hyeondo Jang, Dongyeop Lee, Dan Alistarh, and Namhoon Lee. The unseen frontier: Pushing the limits of llm sparsity with surrogate-free admm. *arXiv preprint arXiv:2510.01650*, 2025.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Jesse Mu and Jacob Andreas. Compositional explanations of neurons. *Advances in Neural Information Processing Systems*, 33:17153–17163, 2020.

Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Bruno A Olshausen and David J Field. Sparse coding of sensory inputs. *Current opinion in neurobiology*, 14(4):481–487, 2004.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Jeff Pool, Abhishek Sawarkar, and Jay Rodge. Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt. *NVIDIA Developer Technical Blog, https://developer. nvidia. com/blog/accelerating-inference-with-sparsityusing-ampere-and-tensorrt*, 2021.

Shashata Sawmya, Linghao Kong, Ilia Markov, Dan Alistarh, and Nir N Shavit. Wasserstein distances, neuronal entanglement, and sparsity. In *The Thirteenth International Conference on Learning Representations*, 2025.

David Slate. Letter recognition. *UCI Machine Learning Repository*, 10:C5ZP40, 1991.

Ashwin Srinivasan. Statlog (landsat satellite). *UCI Machine Learning Repository*, 10:C55887, 1993.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Adly Templeton. *Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet*. Anthropic, 2024.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Xin Yu, Thiago Serra, Srikumar Ramalingam, and Shandian Zhe. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *International Conference on Machine Learning*, pp. 25668–25683. PMLR, 2022.

A Appendix

A.1 Dataset Generation Details

We construct a synthetic Boolean dataset based on monotone disjunctive normal form (DNF) formulas, designed to allow controlled analysis of feature-level structure under sparsification.

DNF Structure. For a Boolean formula with m input literals, where $m \in \{16, 32, 48, 64\}$ in all experiments, we partition the literals into $m/4$ disjoint clauses. Each clause consists of exactly four literals, and no literal appears in more than one clause. The resulting formula is therefore a *read-once monotone DNF* of the form

$$f(x) = \bigvee_{j=1}^{m/4} \bigwedge_{\ell \in S_j} x_\ell,$$

where $\{S_j\}$ form a partition of $\{1, \ldots, m\}$ with $|S_j| = 4$ for all j.

Input Sampling. Each input sample $x \in \mathbb{R}^m$ is initialized to zero. For each sample, we uniformly draw an integer k between $m/4$ and $1.5 \cdot (m/4)$, and randomly select k distinct literals to activate. Activated literals are set to a value of 1.5 rather than 1, which we found improves training stability, while all remaining literals remain at zero.

Dataset Size. The total number of training samples is set to $50{,}000 \cdot (m/16)$, and the total number of test samples is set to $10{,}000 \cdot (m/16)$, ensuring that dataset size scales linearly with input dimensionality.

Label Assignment and Balancing. Each generated sample is evaluated under the target DNF. Samples that satisfy at least one clause are labeled positive, while all others are labeled negative. To maintain a balanced dataset, we track the number of positive and negative samples during construction and discard any sample that would cause either class to exceed half of the total dataset size. This procedure yields approximately balanced class proportions without explicit resampling.

A.2 Training Details

All models are trained using the same optimization and batching procedure across datasets and sparsity settings in PyTorch (Paszke et al., 2019). The experiments are small-scale and complete in minutes on commodity GPU or CPU hardware. We run our experiments on an NVIDIA RTX 2080 Ti.

Optimization Setup. Training is performed using the Adam optimizer with default momentum parameters. All datasets are trained with a fixed batch size of 256.

Base Training. All models are trained for 25 epochs with a learning rate of 10^{-3}. This training procedure is applied uniformly across all Boolean and vision datasets considered in the paper.

Fine-Tuning. When fine-tuning is applied after pruning, we continue training the resulting sparse model for an additional 15 epochs using a reduced learning rate of 10^{-4}. Fine-tuning also uses the Adam optimizer with default momentum parameters, and the same batch size as the base training phase.

No additional learning rate schedules, regularization terms, or early stopping criteria are used.

Code Emergence. Across all experimental conditions, the 4P-positive codes arise in tandem with a decrease in training loss (fig. 11).

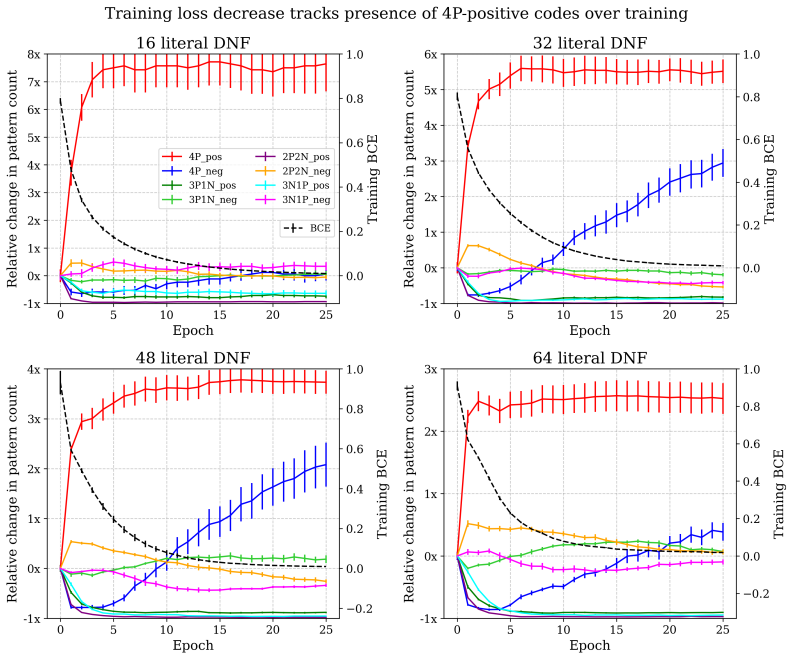

Figure 11: Prevalence of different codes during training. Network performance tracks the prevalence of 4P-positive codes during training. This thus motivates our tracking of such codes over the course of sparsification. 4N codes not shown since their effect after ReLU is zero. Error bars indicate one standard error of the mean. Results collected over ten trials.

A.3 SPARSIFICATION TECHNIQUE DETAILS

We briefly describe the saliency criteria used by the second-order pruning baselines considered in this work, following the classical formulations of Optimal Brain Damage (OBD) (LeCun et al., 1989) and Optimal Brain Surgeon (OBS) (Hassibi & Stork, 1992). Both methods estimate the increase in loss induced by removing individual weights using a local quadratic approximation.

Second-Order Loss Approximation. Let $\mathcal{L}(\mathbf{w})$ denote the training loss as a function of the model parameters $\mathbf{w}$. Around a local minimum $\mathbf{w}$, the loss can be approximated by a second-order Taylor expansion:

$$\mathcal{L}(\mathbf{w} + \Delta\mathbf{w}) \approx \mathcal{L}(\mathbf{w}) + \nabla\mathcal{L}(\mathbf{w})^\top \Delta\mathbf{w} + \tfrac{1}{2}\Delta\mathbf{w}^\top \mathbf{H} \Delta\mathbf{w},$$

where $\mathbf{H}$ is the Hessian of $\mathcal{L}$ with respect to $\mathbf{w}$. At a local optimum, the gradient term vanishes, leaving only the quadratic contribution. In our experiments, we use 10% of the samples in the training dataset to calculate the Hessian for the boolean task and the entire training dataset for the Digits task.

Optimal Brain Damage (OBD). OBD assumes that removing a single weight w_i does not induce compensatory changes in the remaining parameters. Under this assumption, pruning w_i corresponds to the perturbation $\Delta\mathbf{w} = -w_i\mathbf{e}_i$, where $\mathbf{e}_i$ is the i-th standard basis vector. Substituting into the quadratic approximation yields

$$\Delta\mathcal{L}_i \approx \tfrac{1}{2}\mathbf{H}_{ii} w_i^2.$$

This quantity defines the *saliency* of weight w_i. OBD prunes the weights in $\mathbf{W}_1$ with the smallest saliency values, resulting in a curvature-aware refinement of magnitude pruning that accounts for local sensitivity of the loss.

Optimal Brain Surgeon (OBS). OBS relaxes the independence assumption of OBD by allowing the remaining weights to adjust in order to minimize the loss increase caused by pruning. The method considers the constrained optimization problem of removing a weight w_i while optimally updating all other parameters to compensate.

Formally, pruning w_i imposes the constraint $\Delta w_i = -w_i$. Minimizing the quadratic loss approximation subject to this constraint yields the optimal parameter update

$$\Delta\mathbf{w} = -\frac{w_i}{(\mathbf{H}^{-1})_{ii}} \mathbf{H}^{-1}_{\cdot i},$$

and an associated increase in loss given by

$$\Delta\mathcal{L}_i = \tfrac{w_i^2}{2(\mathbf{H}^{-1})_{ii}}.$$

This expression defines the OBS saliency criterion, which explicitly accounts for interactions between parameters through the inverse Hessian. In practice, a damped or approximate inverse Hessian is used for numerical stability.

OBS Compensation Update. When pruning a set of weights S simultaneously, OBS applies the compensation update in closed form. Let $\mathbf{w}_S$ denote the subvector of parameters indexed by S. The optimal update to the remaining parameters is

$$\Delta\mathbf{w} = -\mathbf{H}^{-1}_{\cdot S} (\mathbf{H}^{-1})^{-1}_{SS} \mathbf{w}_S,$$

which minimizes the second-order loss increase subject to setting all pruned weights to zero. This update is applied once after mask selection, yielding a sparsified model that preserves performance more effectively than uncorrected pruning. Note that this update is independent of the choice of mask, which allows us to use an OBS-style weight compensation update with the MIP mask as our FLO condition.

A.4 MINIMIZING DIVERGENCE FROM $\mathbf{C}_1$

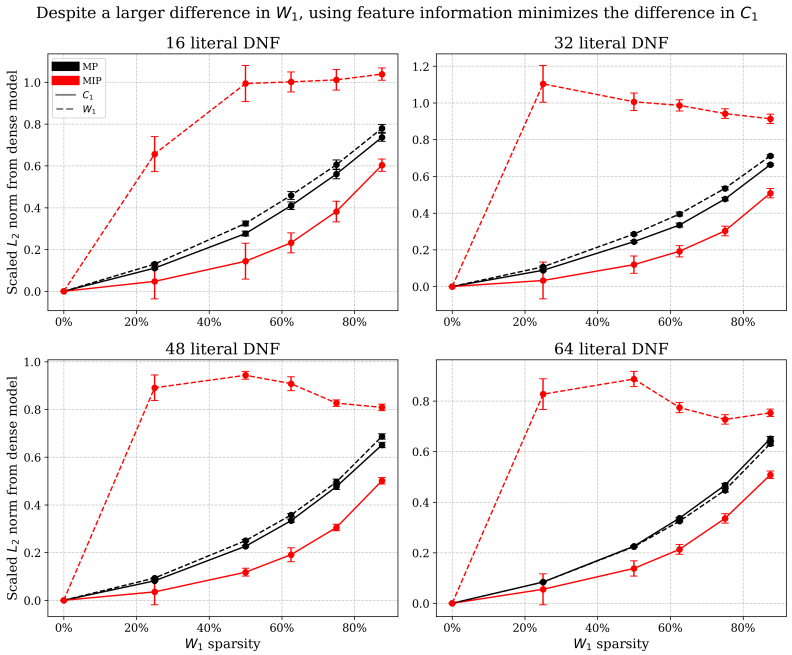

Figure 12: Divergence of $\mathbf{W}_1$ and $\mathbf{C}_1$. Despite MIP inducing a greater relative divergence in $\mathbf{W}_1$ from the dense model compared to magnitude pruning, it induces a lesser relative divergence in $\mathbf{C}_1$. Error bars indicate one standard error of the mean. Results collected over ten trials.

A.5 MASK DIFFERENCES BETWEEN PRUNING TECHNIQUES

After pruning the networks trained on the DNF task, each pruning technique optimizes for a different sparse mask.

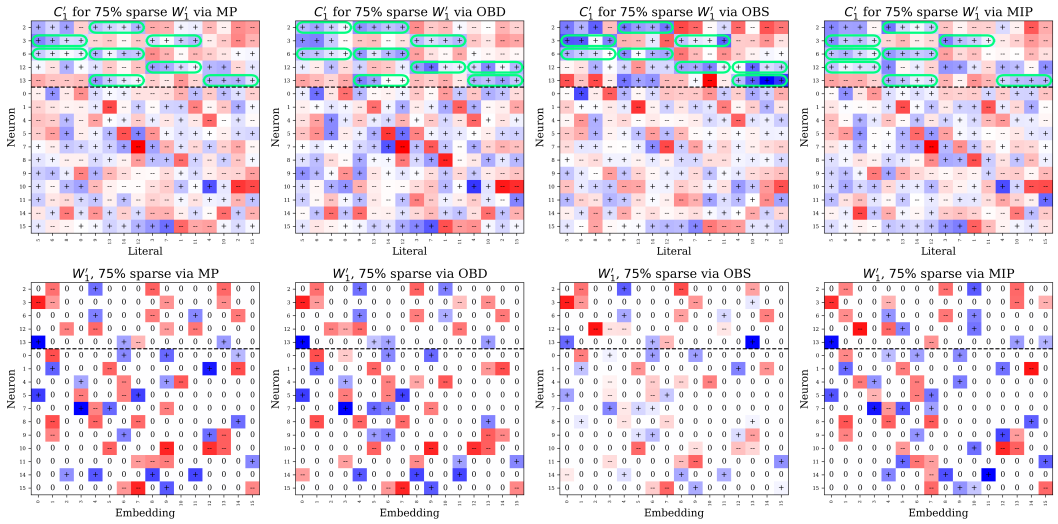

Figure 13: MP, OBD, OBS, and MIP optimize their masks differently. Dense model is same as fig. 1.

A.6 FEATURE-SPACE INFORMATION IMPROVES PRUNING PERFORMANCE

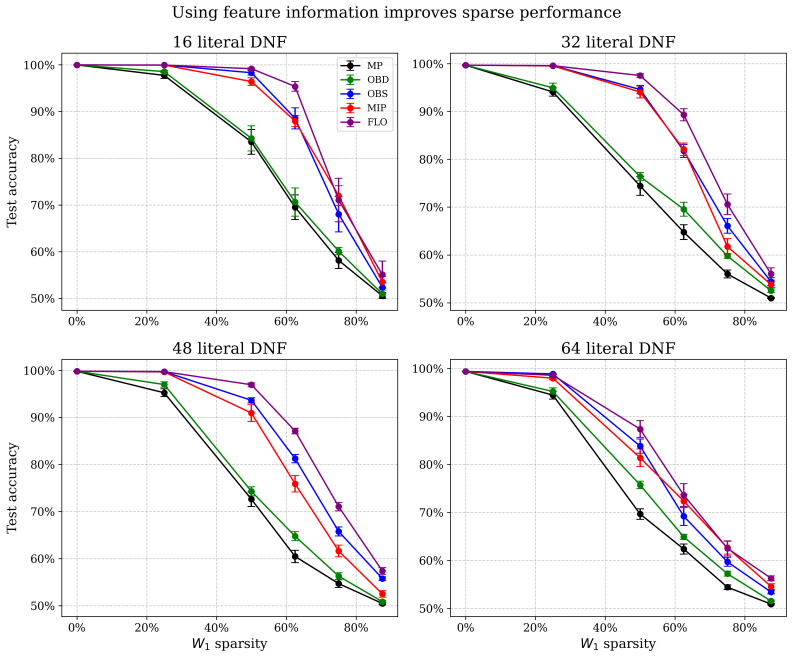

Figure 14: Feature-space information improves performance. Error bars indicate one standard error of the mean. Results collected over ten trials.

A.7 FEATURE-SPACE METHODS BETTER MAINTAIN RELEVANT CODES

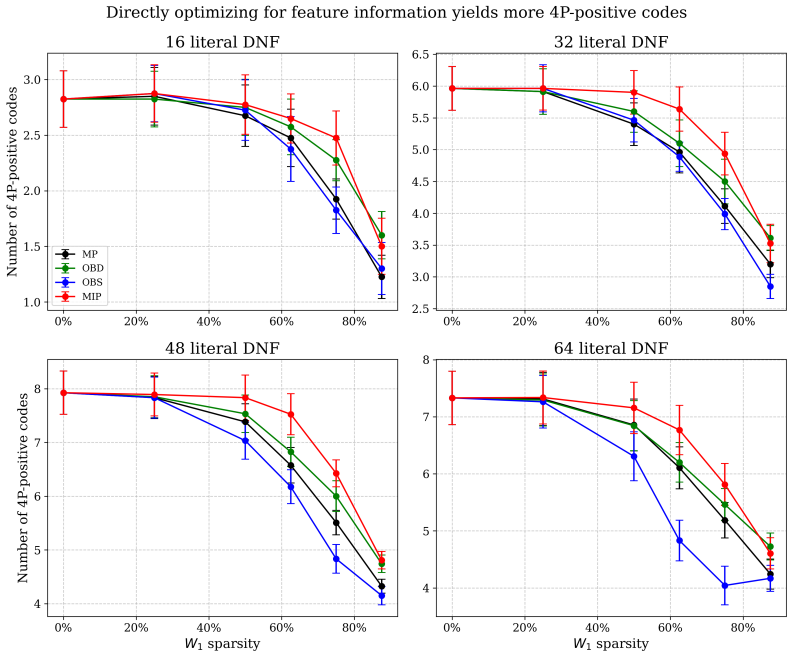

Figure 15: 4P-positive codes are better maintained by MIP. Error bars indicate one standard error of the mean. Results collected over ten trials.

A.8 Extensions to structured sparsity and Digits

We extend our MIP formulation to enforce N:M structured sparsity, a constraint widely used for hardware acceleration where N weights are zero in every block of M contiguous weights (Pool et al., 2021). Specifically, we study $\geq 2:4$ sparsity, where at least 2 of every block of 4 contiguous weights are zero. Even under these constraints, FLO still achieves the highest accuracy, followed by MIP and OBS, with OBD and MP trailing behind. This suggests that the principle of "effective alignment" is robust to the specific granularity of the pruning mask, and that hardware-efficient masks can still support high-fidelity effective weights if selected via the correct objective.

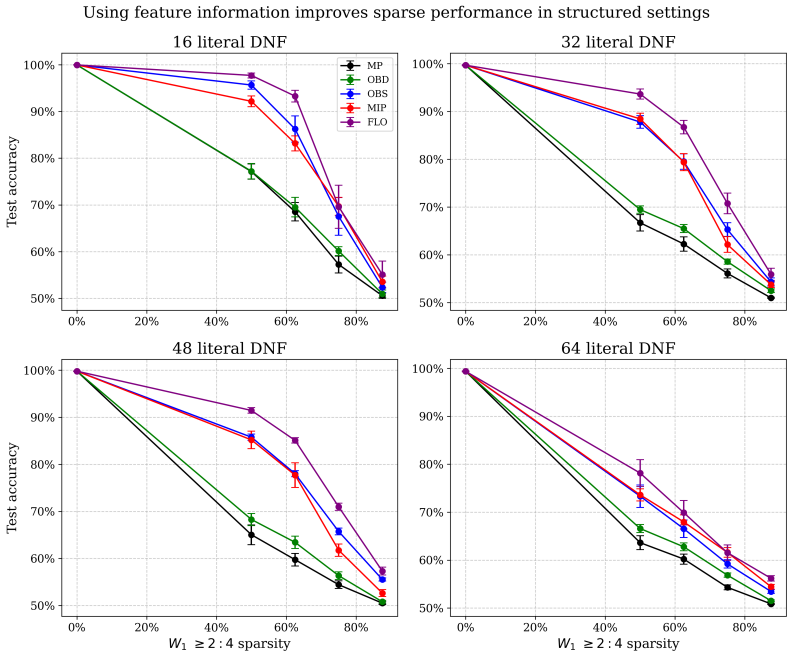

Figure 16: Feature-space information improves sparse performance even in structured settings. Error bars indicate one standard error of the mean. Results collected over ten trials.

We also validate the generality of our hypothesis on the Scikit-learn Digits dataset. Unlike the synthetic DNF task, this dataset involves continuous inputs and requires the extraction of geometric features, and there are no canonical "codes" in feature-space. Even so, consistent with our findings in the boolean domain, feature-aware methods achieve the highest sparse performance, with FLO consistently outperforming all other techniques. In this setting, OBS outperforms MIP, which outperforms OBD and MP. These patterns hold under both unstructured and structured sparsity (fig. 17).

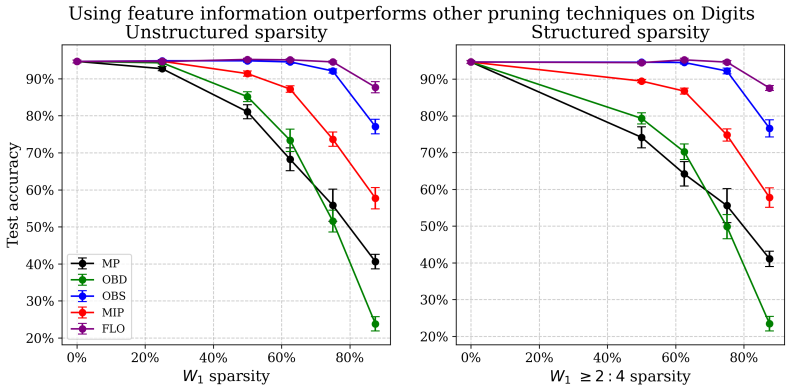

Figure 17: Feature-space information improves performance on the Digits task. Error bars indicate one standard error of the mean. Results collected over ten trials.

This shift in ordering is consistent with the increased importance of weight adaptation in more realistic tasks where small changes in surviving parameters can significantly affect feature quality. Nevertheless, the strong performance of FLO indicates that accurate topological selection remains essential, and that combining feature-aligned masks with weight updates is particularly beneficial in more realistic settings. We also observe that MP, OBD, OBS, and MIP induce qualitatively different masks on digits task, reflecting differences in the features preserved by each method (fig. 18).

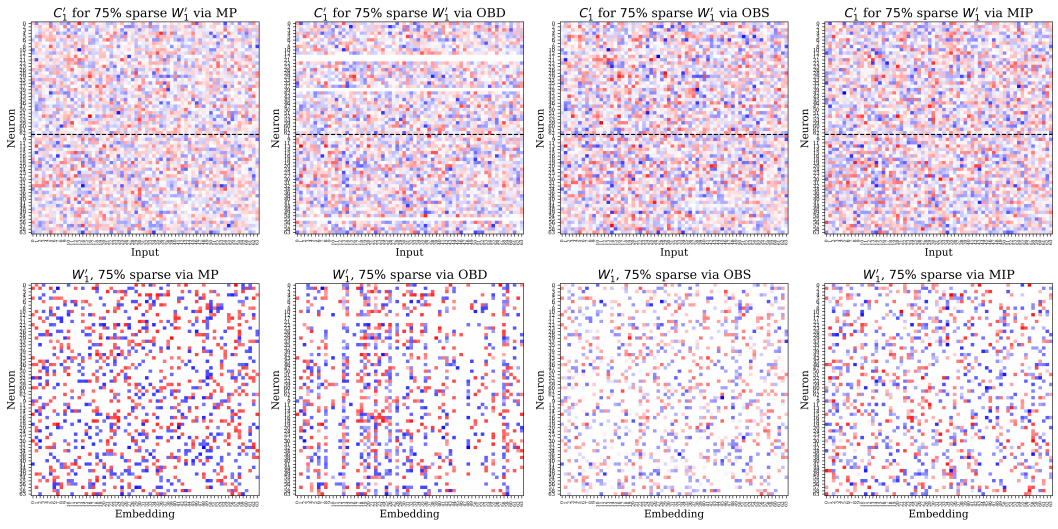

Figure 18: Differences across the pruning techniques for Digits.

A.9 THE IMPACT OF FINETUNING ON CODES

We generally observe that finetuning after pruning maintains the same broad set of codes.

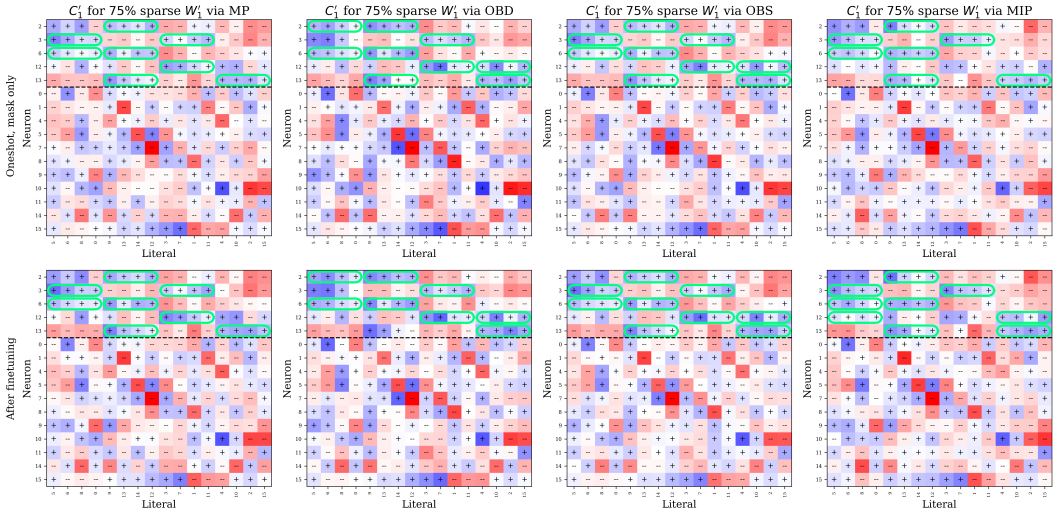

Figure 19: Finetuning largely preserves the same set of codes. Dense model is same as fig. 1.

A.10 ANALYSIS IN DEEPER NETWORKS

To reduce the concern that our controlled architecture is too restrictive, we considered a substantially deeper multi-layer variant in which the network contains five hidden layers prior to the classifier, with repeated learned mixing stages:

$$
\begin{aligned}
h_0 &= \mathbf{C}_0 x \\
h_i &= \mathbf{C}_i(\text{ReLU}(\mathbf{W}_i h_{i-1} + b_i)), \quad i = 1, \ldots, 4 \\
h_5 &= \text{ReLU}(\mathbf{W}_5 h_4 + b_5) \\
z &= \mathbf{W}_6 h_5 + b_6
\end{aligned}
$$

Although this setting is less analytically clean than the original single-layer construction, it relaxes the setup while preserving the core question. We now prune each $\mathbf{W}_i$ layer using MP, OBD, OBS, MIP, and FLO, except for $\mathbf{W}_6$. For each $\mathbf{W}_i$, MIP targets preservation of $\mathbf{W}_i\mathbf{C}_{i-1}$, while other methods use their respective saliencies.

For 16 literals, a 16 dimensional $\mathbf{C}_{i-1}$ output, and a $\mathbf{W}_i$ hidden size of 16:

	Sparsity			
Method	50%	62.5%	75%	87.5%
MP	65.0±2.8	57.5±1.4	50.5±1.3	48.6±0.4
OBD	62.5±3.2	52.9±1.7	49.2±1.1	50.2±0.9
OBS	83.6±2.5	59.4±2.0	48.5±1.7	49.2±0.1
MIP	84.0±3.0	**68.8±3.1**	**60.0±2.2**	**51.1±1.9**
FLO	**90.8±1.5**	67.3±4.3	51.4±1.2	49.3±0.0

Table 1: Pruning technique performance in deeper network. Error bars indicate one standard error of the mean. Results collected over ten trials.

Feature-aware methods remain strongest overall in this deeper architecture: FLO performs best at 50% sparsity, while MIP performs best at higher sparsities. We view this as evidence that the observed performance differences are not confined to the narrowest, most directly interpretable version of the original setup, but can persist in a substantially deeper controlled architecture as well.

A.11 SPARSE AUTOENCODER TRAINING DETAILS

Training data. For each trained dense model, we collect the outputs $\mathbf{x}' = \mathbf{C}_0\mathbf{x}$ of the embedding layer and train the SAE only on these activations, without labels or access to $\mathbf{C}_0$ during optimization. For the Letter Recognition, Pendigits, Satimage, and Waveform datasets, we use the embedding outputs of the training samples. A separate SAE is trained for each model trial.

Architecture and objective. We use the Top-K SAE described in section 4.4. For an activation $\mathbf{x}'$, the encoder first computes $\mathbf{a} = \mathbf{W}_{\text{enc}}(\mathbf{x}' - \mathbf{b}_{\text{dec}}) + \mathbf{b}_{\text{enc}}$, selects its K largest preactivations, and then applies ReLU to the selected values. The resulting code therefore has at most K nonzero entries. The decoder reconstructs $\hat{\mathbf{x}}' = \mathbf{W}_{\text{dec}}\mathbf{z} + \mathbf{b}_{\text{dec}}$. We initialize $\mathbf{W}_{\text{enc}}$ with Kaiming uniform initialization, initialize the decoder from the transpose of the encoder, and initialize $\mathbf{b}_{\text{dec}}$ to the mean training activation. Decoder atoms are constrained to unit norm by renormalizing them after every optimization step.

AuxK loss and optimization. The primary objective is mean squared reconstruction error. We additionally use the AuxK loss of Gao et al. (2024) to reduce dead latents. A latent is considered dead if it has not been selected among the Top-K preactivations for 400 optimizer steps. When dead latents are present, we select up to K of them to reconstruct the detached residual $\mathbf{x}' - \hat{\mathbf{x}}'$ and add this auxiliary reconstruction loss with coefficient $1/32$. We train with Adam for 200 epochs using a learning rate of 10^{-3} and batch size 1024, shuffling the activation samples each epoch. We use the final-epoch SAE without early stopping or validation-based checkpoint selection.

Configurations and pruning. For the four benchmark datasets in fig. 10, we fix a single representative configuration with latent width $m = 32$ and $K = 6$. This intermediate-capacity setting balances dictionary width, latent sparsity, and reconstruction quality; importantly, it is held fixed across datasets and weight sparsities rather than tuned separately for each condition. To expose sensitivity to this choice, we also report a full sweep over $m \in \{8, 16, 24, 32, 48\}$ and $K \in \{2, 4, 6, 8, 12, 16\}$, restricted to $K \leq m$. After training, the decoder matrix is used as the surrogate feature dictionary in the MIP objective. We use only the decoder atoms: the decoder bias is omitted and no corresponding compensation is applied. We do not align or rescale this dictionary using the true $\mathbf{C}_0$. The surrogate is used only to select and optimize the sparse $\mathbf{W}_1$ topology; the deployed network retains its original embedding layer. MIP-SAE applies the resulting sparse weights and mask directly, while FLO-SAE applies the same mask followed by the OBS compensation update. Reconstruction error is evaluated on held-out embedding activations, and pruning performance is evaluated at 50%, 62.5%, 75%, and 87.5% sparsity.

Sensitivity to dictionary capacity and code sparsity. The sweeps in figs. 20 to 23 reveal a capacity–selectivity trade-off. The dictionary width m must be sufficiently large to represent the activation space, while K must permit enough simultaneously active features to reconstruct each activation. When m or K is too small, the SAE forms an information bottleneck and reconstruction or pruning performance can degrade. Conversely, overly large K weakens code sparsity and permits reconstruction to be distributed across less selective or redundant atoms, which can reduce the feature alignment useful for pruning even when reconstruction error improves. Thus, reconstruction fidelity and preservation of pruning-relevant directions are related but not identical, and effective feature-aware pruning requires adequate dictionary capacity together with an intermediate level of code sparsity. The fixed choice $m = 32$, $K = 6$ represents this intermediate regime without condition-specific tuning.

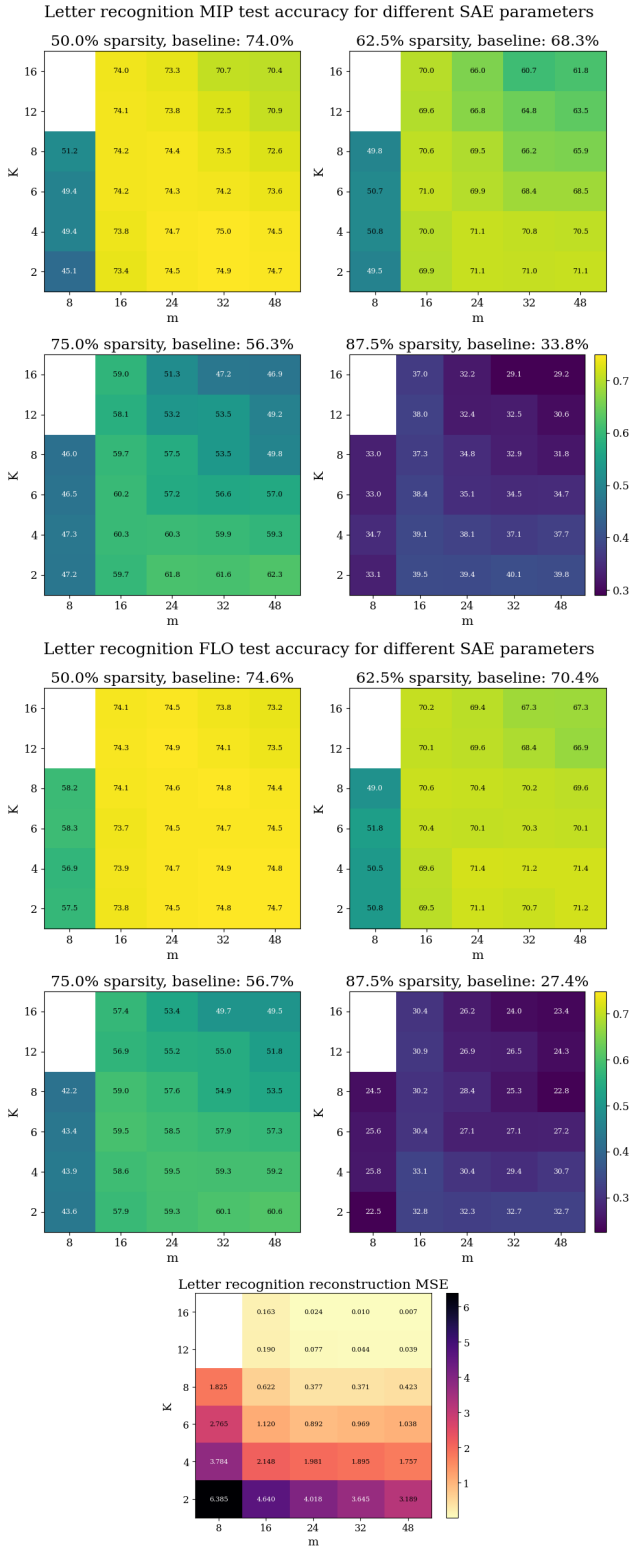

Figure 20: SAE hyperparameter sensitivity on Letter Recognition. MIP-SAE and FLO-SAE test accuracy across latent width m, active-latent count K, and weight sparsity; panel titles report the corresponding oracle $\mathbf{C}_0$ baseline test accuracy. The final heatmap shows held-out activation reconstruction MSE. Blank cells denote $K > m$. Results collected over ten trials.

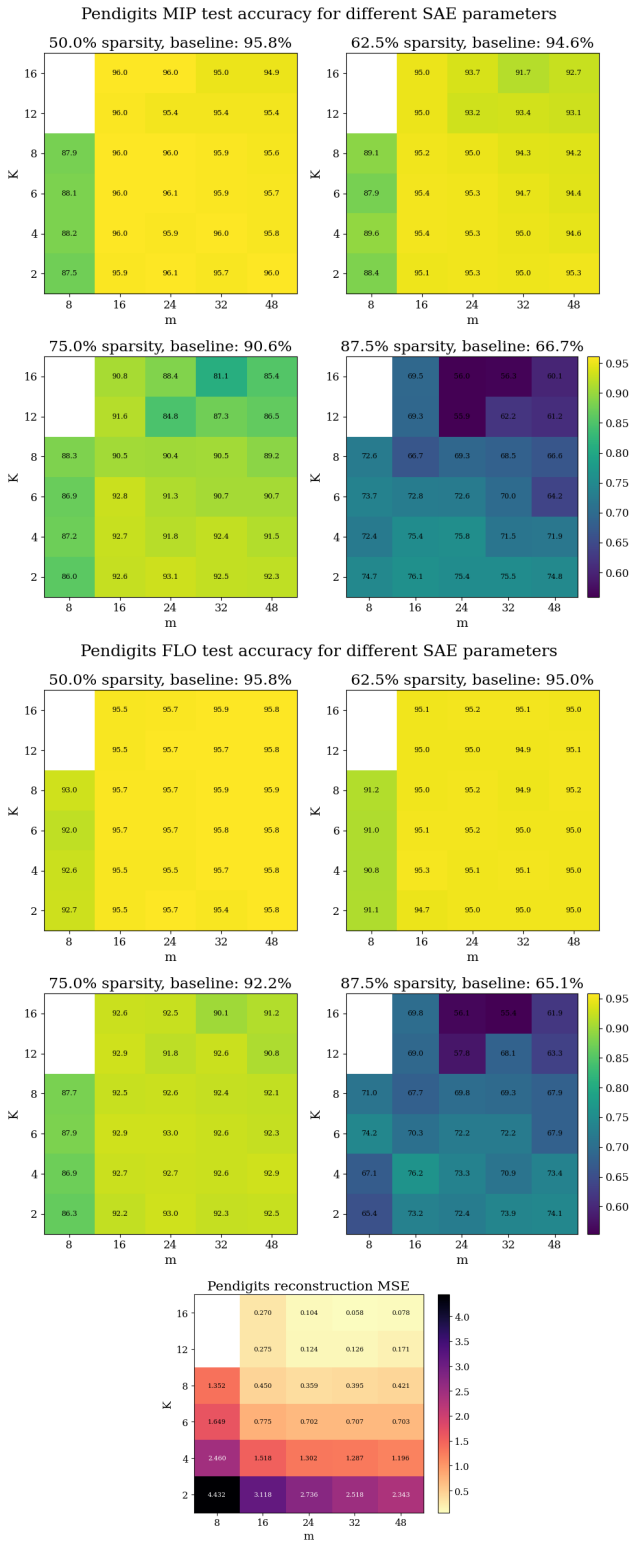

Figure 21: SAE hyperparameter sensitivity on Pendigits. MIP-SAE and FLO-SAE test accuracy across latent width m, active-latent count K, and weight sparsity; panel titles report the corresponding oracle $\mathbf{C}_0$ baseline test accuracy. The final heatmap shows held-out activation reconstruction MSE. Blank cells denote $K > m$. Results collected over ten trials.

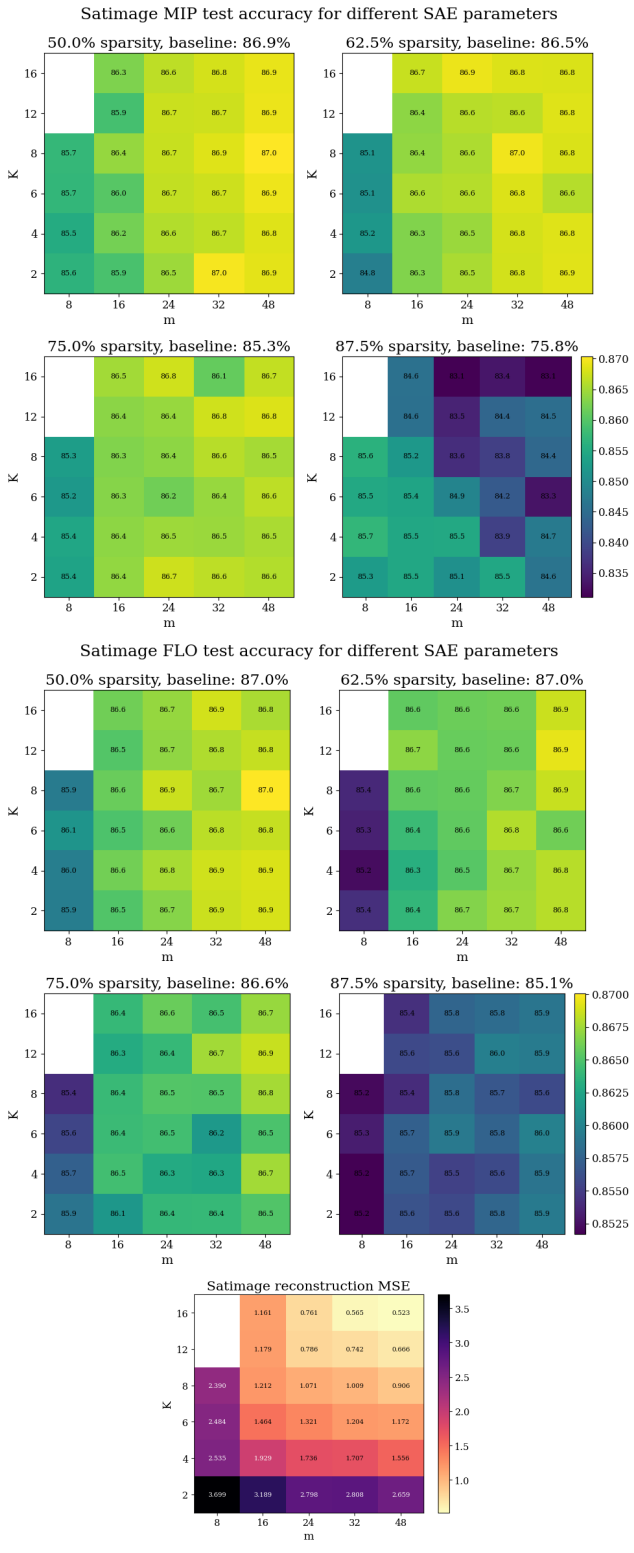

Figure 22: SAE hyperparameter sensitivity on Satimage. MIP-SAE and FLO-SAE test accuracy across latent width m, active-latent count K, and weight sparsity; panel titles report the corresponding oracle $\mathbf{C}_0$ baseline test accuracy. The final heatmap shows held-out activation reconstruction MSE. Blank cells denote $K > m$. Results collected over ten trials.

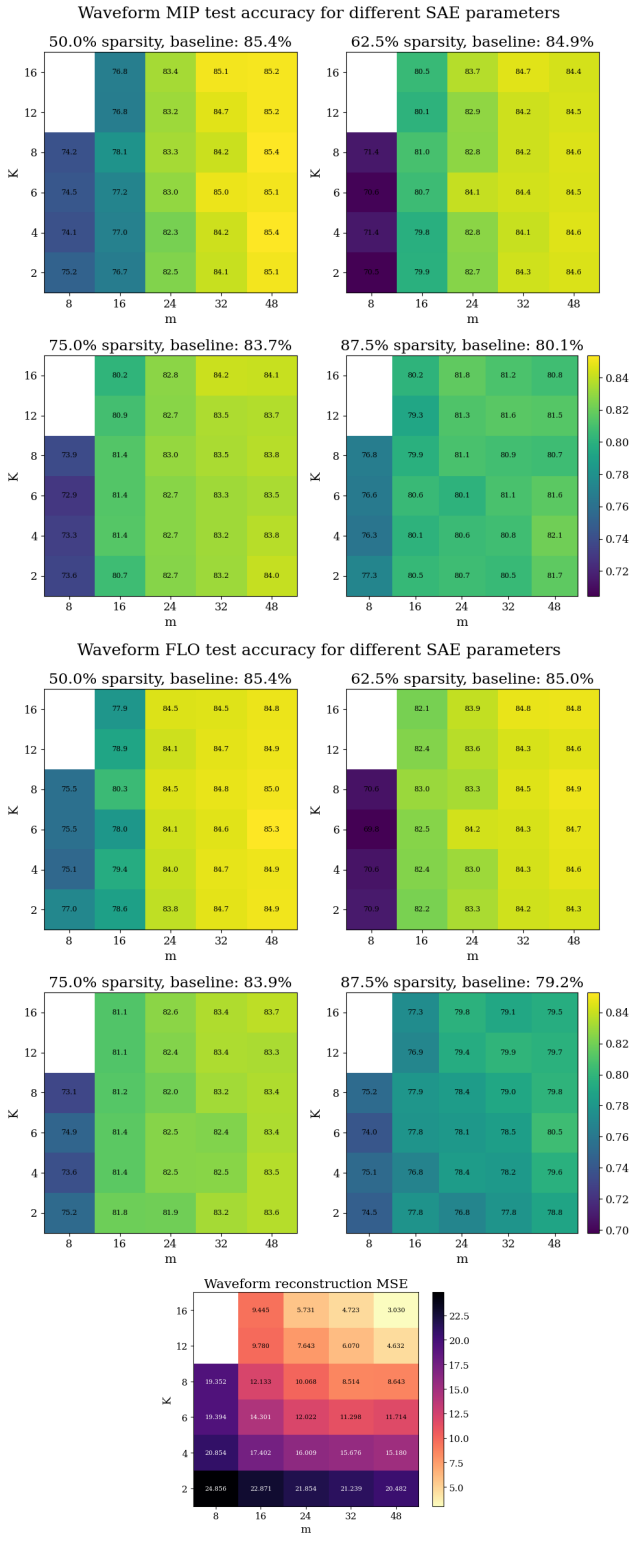

Figure 23: SAE hyperparameter sensitivity on Waveform. MIP-SAE and FLO-SAE test accuracy across latent width m, active-latent count K, and weight sparsity; panel titles report the corresponding oracle $\mathbf{C}_0$ baseline test accuracy. The final heatmap shows held-out activation reconstruction MSE. Blank cells denote $K > m$. Results collected over ten trials.