SPRBM: Adaptive Parameter-Wise Pruning via Task–Complexity Gradient Alignment

Anonymous authors
Paper under double-blind review

Abstract

Parameter importance in neural-network pruning is commonly estimated from static weight statistics, predefined sparsity rules, or isolated gradient snapshots. Such criteria do not directly control how strongly each parameter is regularized during task optimization. We introduce SPRBM, a sparsification framework that uses task–complexity gradient agreement for parameter-wise adaptive regularization during training and post-training pressure calibration. For every trainable parameter, SPRBM measures the signed agreement between the task gradient and the gradient of a differentiable complexity objective. This signal determines an adaptive regularization coefficient during training. After training, coefficients recomputed on multiple validation mini-batches are averaged and converted into parameter-specific pruning thresholds. The resulting framework does not impose a fixed global sparsity pattern. Instead, pruning pressure emerges from the interaction between task optimization, parameter complexity, and post-training validation-batch calibration. We instantiate SPRBM with L_1 and L_2 complexity functions and evaluate it on autoencoders, restricted Boltzmann machines, vision transformers, and LoRA adapter sparsification for language adaptation. Across these settings, SPRBM produces architecture-dependent sparsification regimes, including approximately 85% sparsity for the RBM, 30–35% sparsity for vision transformers, and 38–53% sparsification of LoRA parameters. Matched-sparsity ablations further show that disrupting the correspondence between the learned alignment signal and parameter locations degrades performance, indicating that the observed behavior cannot be explained by sparsity level alone.

1 Introduction

Modern neural networks are heavily over-parameterized, creating opportunities for compression through pruning. The central difficulty is not merely deciding how many parameters to remove, but determining which parameters can tolerate increasing sparsification pressure without substantially interfering with task optimization. Many pruning strategies treat parameter importance as a largely static quantity. Magnitude pruning associates small parameter magnitude with low importance, while gradient-based criteria often measure sensitivity at initialization or at a limited number of training snapshots. Fixed regularization approaches similarly encourage sparsity using global penalties. These strategies assume that parameter utility can be summarized by a fixed statistic. However, a parameter's local relationship with the task objective also determines whether complexity reduction assists or interferes with the current task descent direction. This motivates our central question:

> *Can parameter-wise pruning pressure be derived from task–complexity gradient agreement rather than assigned from a static importance criterion?*

We address this question with SPRBM (Synergistic Pruning with Adaptive Regularization via Gradient Alignment), a training-time adaptive sparsification framework. Rather than directly predicting whether a parameter is "important" or "redundant," SPRBM learns how strongly each parameter can be regularized according to the interaction between the task objective and a differentiable complexity objective. For each trainable parameter, we compare

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.

the task gradient with the complexity gradient. Their signed agreement indicates whether a local update toward lower complexity is compatible with the current task descent direction. This signal determines a parameter-specific regularization coefficient. After optimization, SPRBM recomputes these coefficients on multiple validation mini-batches and derives parameter-specific thresholds from their average. Thus, SPRBM should not be interpreted as directly estimating an immutable parameter-importance score. It estimates *pruning pressure*: the extent to which complexity reduction is locally compatible with task descent across the post-training calibration batches. This mechanism allows architectures, layers, and parameter groups to converge to different sparsity regimes. We therefore evaluate both final performance and the behavior of the learned sparsification signal. Our contributions are:

- We introduce SPRBM, a training-time adaptive sparsification framework that learns parameter-specific pruning pressure from parameter-wise task–complexity gradient interaction and converts this signal into adaptive regularization.
- We provide a first-order interpretation showing that the sign of task–complexity gradient agreement determines whether additional complexity pressure locally assists or interferes with task descent.
- We use post-training validation-batch-averaged pressure to construct parameter-specific thresholds that produce different sparsification regimes across the evaluated autoencoders, RBMs, vision transformers, and LoRA adaptation parameters.
- Through matched-sparsity random, constant, shuffled, and magnitude ablations, we show that preserving the correspondence between the learned signal and its parameter location matters beyond sparsity level alone.

2 Related Work

Neural-network sparsification differs primarily in the signal used to select parameters and when that signal is measured.

Static and initialization-time criteria. Magnitude pruning removes small weights and remains a standard compression baseline (Han et al., 2015; Frankle & Carbin, 2019). Initialization-time methods such as SNIP and GraSP instead use task-gradient sensitivity or gradient-flow preservation (Lee et al., 2019; Wang et al., 2020). These criteria are useful baselines but do not combine parameter-wise task–complexity agreement with adaptive regularization and post-training pressure calibration.

Training-time and history-aware pruning. Sparse connectivity can be modified throughout optimization (Bellec et al., 2018; Mostafa & Wang, 2019; Evci et al., 2020). More directly, Movement Pruning accumulates signed weight–gradient movement during fine-tuning, while PLATON temporally smooths first-order sensitivity and its uncertainty (Sanh et al., 2020; Zhang et al., 2022). For the L_2 complexity instantiation, the unnormalized task–complexity interaction is related to the weight-gradient product used in movement-based approaches. Unlike Movement Pruning and PLATON, SPRBM does not temporally accumulate or smooth a score across optimizer steps. It instead differs in three aspects: (1) parameter-wise normalization of task–complexity agreement, (2) transformation of this signal into an adaptive regularization coefficient, and (3) construction of parameter-specific pruning thresholds from coefficients averaged across post-training validation mini-batches.

Regularization, learned masks, and gradient interaction. Regularization-based methods learn sparse solutions through global, group-specific, or learned sparsity mechanisms, including L_0 regularization and growing regularization (Louizos et al., 2018; Wang et al., 2021). Recent pruning work also uses interactions between multiple task gradients: GAPrune combines cross-domain gradient alignment with importance, while Domain-Aware Pruning smooths cross-domain gradient-sign agreement to influence learned masks (Tang & Yang, 2025; Sazdar et al., 2026). These methods establish gradient agreement as a pruning signal

Table 1: Conceptual comparison of representative pruning signals.

Method	Primary signal	Measurement regime
Magnitude	$\lvert\theta_i\rvert$	Current/final weights
SNIP/GraSP	Task-gradient sensitivity	Initialization
Movement/PLATON	Task-gradient-derived score	Training history
SPRBM	Task–complexity agreement	Training + post-training calibration

in multi-domain settings; SPRBM instead measures parameter-wise agreement between the task gradient and the gradient of a differentiable complexity objective.

Parameter-efficient adaptation and positioning. Adaptive low-rank methods such as AdaLoRA use temporally smoothed gradient-derived importance to allocate rank budgets during fine-tuning (Zhang et al., 2023). We use LoRA only as a test of unstructured sparsification within trainable adapter parameters, not as evidence for full-model LLM pruning. Table 1 summarizes the closest signal families.

Prior work therefore separately establishes gradient-derived importance, history-aware pruning, adaptive sparsification, and gradient agreement. To the best of our knowledge, we are not aware of prior pruning methods that combine parameter-wise task–complexity gradient agreement, adaptive regularization during training, post-training validation-batch pressure calibration, and parameter-specific survival thresholds in a single framework.

3 Method

SPRBM estimates parameter-wise task–complexity gradient agreement, converts it into adaptive regularization coefficients during training, and uses post-training validation-batch-averaged coefficients to construct parameter-specific pruning thresholds.

3.1 Problem Formulation

Consider trainable parameters

$$\theta = \{\theta_i\}_{i=1}^{N}, \tag{1}$$

optimized for the task objective

$$\min_{\theta} \mathcal{L}_{\text{task}}(\theta). \tag{2}$$

We additionally define a separable differentiable complexity objective

$$\mathcal{R}(\theta) = \sum_{i=1}^{N} \mathcal{C}(\theta_i). \tag{3}$$

We study two instantiations:

$$\mathcal{C}_{L_1}(\theta_i) = |\theta_i|, \tag{4}$$

and

$$\mathcal{C}_{L_2}(\theta_i) = \frac{1}{2}\theta_i^2. \tag{5}$$

For L_1, the derivative at zero can be implemented using a standard subgradient convention. The formulation permits other suitable differentiable complexity functions; our experiments evaluate only the L_1 and L_2 instantiations.

3.2 Parameter-Wise Gradient Agreement

For parameter θ_i, define

$$g_i^{\text{task}} = \frac{\partial \mathcal{L}_{\text{task}}}{\partial \theta_i}, \tag{6}$$

and

$$g_i^{\text{comp}} = \frac{\partial \mathcal{C}(\theta_i)}{\partial \theta_i}. \tag{7}$$

We compute

$$\gamma_i^{(t)} = \frac{g_i^{\text{task}} g_i^{\text{comp}}}{|g_i^{\text{task}}||g_i^{\text{comp}}| + \epsilon}, \tag{8}$$

where t is the training step and $\epsilon > 0$ stabilizes the ratio. Because eq. (8) is evaluated parameter-wise, γ_i is a bounded *signed gradient-agreement signal*, rather than a high-dimensional cosine similarity. Its sign records whether task descent and complexity descent favor compatible parameter directions. If

$$g_i^{\text{task}} g_i^{\text{comp}} > 0, \tag{9}$$

the negative task-gradient and negative complexity-gradient updates move θ_i in the same local direction. Conversely,

$$g_i^{\text{task}} g_i^{\text{comp}} < 0 \tag{10}$$

indicates local disagreement, where stronger complexity reduction opposes the current task descent direction. Thus alignment does not prove global redundancy. It measures whether reducing parameter complexity is locally compatible with the current task-optimization state.

3.3 First-Order Interpretation

The connection between gradient agreement and task interference follows from a local expansion.

Proposition 1. *Consider the update*

$$\theta_i' = \theta_i - \rho \left(g_i^{\text{task}} + q_i g_i^{\text{comp}} \right), \tag{11}$$

where $\rho > 0$ is sufficiently small and $q_i \geq 0$ is effective complexity pressure. To first order, the complexity term changes the task-loss decrease by

$$-\rho q_i g_i^{\text{task}} g_i^{\text{comp}}. \tag{12}$$

Thus positive agreement assists local task descent, whereas negative agreement reduces the predicted task-loss decrease.

Proof. Using a first-order Taylor expansion,

$$\mathcal{L}_{\text{task}}(\theta') - \mathcal{L}_{\text{task}}(\theta) \approx \sum_i g_i^{\text{task}}(\theta'_i - \theta_i) \tag{13}$$

$$= -\rho \sum_i \left[(g_i^{\text{task}})^2 + q_i g_i^{\text{task}} g_i^{\text{comp}} \right]. \tag{14}$$

The first term is standard task-gradient descent; the second is generated by complexity regularization and is controlled by the sign of $g_i^{\text{task}} g_i^{\text{comp}}$. □

This result is a local optimization interpretation, not a global guarantee of final pruning quality.

3.4 Adaptive Regularization

We transform the agreement signal into a parameter-specific coefficient:

$$\lambda_i^{(t)} = \Lambda_t \sigma\left(\alpha \gamma_i^{(t)}\right), \tag{15}$$

where Λ_t controls the global scale, $\sigma(\cdot)$ is the sigmoid function, and α determines alignment sensitivity. The training objective is

$$\mathcal{L}_{\text{SPRBM}} = \mathcal{L}_{\text{task}} + \beta \sum_i \lambda_i^{(t)} \mathcal{C}(\theta_i). \tag{16}$$

Since $\lambda_i^{(t)}$ depends on the task gradient through the agreement signal, directly differentiating through this coefficient would introduce second-order optimization terms. In the SPRBM optimization procedure, $\lambda_i^{(t)}$ is treated as a detached adaptive coefficient during the backward pass. Therefore, gradient flow through the pressure estimation process is stopped.

3.5 Adaptive Scaling

Task-gradient magnitudes vary across architectures and training stages. We use

$$\beta = \eta \frac{\overline{|g^{\text{task}}|}}{\Lambda_{\max}}, \tag{17}$$

where η is a normalization coefficient and $\overline{|g^{\text{task}}|}$ denotes the global mean absolute task-gradient magnitude computed over all trainable prunable parameters and the first $M = 20$ training mini-batches. This calibration is performed once before training; β is then held fixed. The implementation captures and restores the Python, NumPy, CPU PyTorch, and CUDA random-number-generator states around this pass so that calibration does not shift the stochastic training stream. This scaling reduces direct sensitivity to absolute gradient scale, although it does not eliminate all hyperparameter dependence.

3.6 Post-Training Pruning-Pressure Calibration

Let θ_T denote the final trained parameters. After training, the implementation places the model in training mode and processes the first $K = 20$ mini-batches of the validation split without optimizer updates. Thus, stochastic layers, when present, remain active during calibration. For validation mini-batch b, it recomputes the task gradient and defines

$$\lambda_i^{(b,\text{cal})} = \Lambda_{\max} \sigma\Big(\alpha \gamma_i^{(b,\text{cal})}(\theta_T)\Big). \tag{18}$$

The final pressure statistic is the validation-batch average

$$\bar{\lambda}_i = \frac{1}{K} \sum_{b=1}^{K} \lambda_i^{(b,\mathrm{cal})}. \tag{19}$$

We interpret $\bar{\lambda}_i$ as post-training validation-batch-averaged pruning pressure. Agreement across calibration batches produces stronger pressure, whereas disagreement produces weaker pressure. Although the scalar agreement is largely sign-like away from near-zero gradients, averaging across mini-batches yields a non-binary statistic. This procedure aggregates evidence across data batches at the terminal model; it does not aggregate a temporal history of optimizer steps.

3.7 Parameter-Specific Pruning Decisions

Global magnitude pruning can be written as

$$m_i = \mathbb{I}\left(|\theta_i| > \tau\right). \tag{20}$$

SPRBM instead defines

$$\tau_i = \tau_{\min} + \left(\tau_{\max} - \tau_{\min}\right) \bar{\lambda}_i. \tag{21}$$

The final mask is

$$m_i = \mathbb{I}\left(|\theta_i| \geq \tau_i\right). \tag{22}$$

Magnitude therefore remains part of the survival decision, but its threshold depends on post-training calibrated pressure. Because $0 \leq \bar{\lambda}_i \leq \Lambda_{\max} = 0.2$, the primary runs use an effective threshold range of approximately 0.005 to 0.014 under eq. (21); $\tau_{\max} = 0.05$ parameterizes the affine mapping but is not the realized upper threshold in the unscaled primary procedure.

3.8 Overall Procedure

In the evaluated vision transformers, SPRBM tends to assign stronger sparsification to feed-forward components than to attention projections. LoRA-A and LoRA-B parameters also receive different sparsity levels. These observations are consistent with parameter-specific pressure, but are not claimed as universal architectural redundancy patterns. Detailed layer-wise results are provided in the appendix.

3.9 LoRA-Based Language Adaptation

We evaluate whether the same mechanism can sparsify the trainable LoRA parameter space of DistilGPT2.

SPRBM-L_2 removes 38.44% of LoRA parameters while increasing perplexity from 46.774 to 46.846. At approximately 53% sparsity, SPRBM-L_1 achieves lower perplexity than the tested magnitude, fixed-regularization, WarmStart-SNIP, random, and SNIP-at-initialization alternatives. This supports a specific claim: post-training validation-batch-averaged pruning pressure can sparsify LoRA adapter parameters while preserving competitive language-modeling quality compared with the tested matched-sparsity baselines. It does not establish equivalent behavior for pruning full large-language-model backbones.

Algorithm 1 SPRBM Adaptive Training and Post-Training Pressure Calibration

Require: Parameters θ, task loss $\mathcal{L}_{task}$, complexity function $\mathcal{C}$, calibration sizes M and K

Estimate fixed β from the first M calibration batches
Restore the random-number-generator states used before β calibration
for each training step t **do**
  Compute task gradients: $g_i^{task} = \frac{\partial \mathcal{L}_{task}}{\partial \theta_i}$
  Compute complexity gradients: $g_i^{comp} = \frac{\partial \mathcal{C}(\theta_i)}{\partial \theta_i}$
  Compute parameter-wise gradient agreement: $\gamma_i^{(t)}$
  Convert agreement into adaptive regularization: $\lambda_i^{(t)}$
  Optimize: $\mathcal{L}_{SPRBM} = \mathcal{L}_{task} + \beta \sum_i \lambda_i^{(t)} \mathcal{C}(\theta_i)$
end for
Let $\theta_T \leftarrow \theta$ and initialize $\bar{\lambda}_i = 0$
Place the model in training mode for post-training calibration
for each of the first K validation mini-batches b **do**
  Compute task and complexity gradients at θ_T without an optimizer update
  Compute $\gamma_i^{(b,\text{cal})}(\theta_T)$ and $\lambda_i^{(b,\text{cal})} = \Lambda_{\max}\sigma(\alpha\gamma_i^{(b,\text{cal})})$
  Accumulate $\lambda_i^{(b,\text{cal})}$ into $\bar{\lambda}_i$
end for
Divide each accumulated $\bar{\lambda}_i$ by K
Compute parameter-specific thresholds: $\tau_i = \tau_{min} + (\tau_{max} - \tau_{min})\bar{\lambda}_i$
Generate final mask: $m_i = \mathbb{I}(|\theta_i| \geq \tau_i)$
return Sparse model with mask m

Table 2: DistilGPT2 LoRA adaptation. Sparsity refers to adapter parameters, not the frozen backbone. Lower perplexity is better.

Method	Perplexity	Adapter sparsity	Remaining
Dense	46.774	0%	147456
SPRBM-L_2	46.846	38.44%	90781
SPRBM-L_1	47.139	52.97%	69355
Magnitude	47.397	53%	69355
Fixed-L_2	47.401	≈53%	69355
Fixed-L_1	47.437	≈53%	69355
WarmStart-SNIP	49.320	≈53%	69355
Random	51.128	53%	69355
SNIP-at-init	55.862	53%	69355

3.10 Computational Trade-Off

SPRBM introduces additional computation because task and complexity signals are evaluated during optimization and again during post-training calibration. The additional memory requirement is linear in the number of prunable parameters, since parameter-wise adaptive coefficients are formed during training and a parameter-sized pressure accumulator is maintained during post-training calibration. Therefore, the additional memory complexity is $O(N)$ for N prunable parameters. This overhead is introduced to obtain parameter-specific thresholds rather than relying on a single static post-training score. For unstructured pruning, parameter sparsity does not automatically translate into proportional wall-clock acceleration on dense hardware. For LoRA, the clearest direct benefit is reduced adapter storage rather than reduced frozen-backbone computation.

4 Discussion

The experiments distinguish *static importance* from *gradient-alignment-guided pruning pressure*. SPRBM does not claim that a parameter with large $\bar{\lambda}_i$ is functionally irrelevant under every perturbation; it records that stronger complexity pressure is comparatively compatible

with task descent across the post-training validation calibration batches. Averaging across calibration batches reduces reliance on a single local gradient product, while matched-sparsity ablations show that the learned signal assignment contains information beyond final sparsity. Architecture-dependent results further show that the framework does not enforce a universal pruning ratio. Overall, the evidence supports the focused conclusion that task–complexity gradient agreement provides a useful parameter-specific pruning signal and that preserving the signal–parameter relationship can improve decisions relative to the tested alternatives. Because no direct comparison with recent trajectory-aware pruning methods is included, our empirical claims are restricted to the baselines evaluated here.

5 Limitations

The final pressure statistic is estimated at the terminal trained model and averaged across validation mini-batches. The current implementation does not retain a temporal history of adaptive coefficients throughout optimization; the experiments therefore support batch-aggregated terminal pressure rather than trajectory-level aggregation. The final mask also depends on the selected validation mini-batches and on stochastic-layer behavior during calibration. The theoretical analysis is local and first-order and does not guarantee that the final pruning mask minimizes post-pruning loss. Our experimental comparisons are restricted to magnitude, fixed regularization, random, and SNIP-style criteria; we therefore do not claim empirical superiority over trajectory-aware methods such as Movement Pruning or PLATON. Direct comparisons with these methods remain an important direction for future evaluation. The present experiments also use unstructured sparsification, so parameter reduction should not be equated with proportional hardware acceleration. Finally, the language-model experiment sparsifies only DistilGPT2 LoRA parameters and does not support claims about full-model LLM pruning.

6 Conclusion

We introduced SPRBM, a training-time adaptive sparsification framework that uses signed task–complexity gradient agreement to control parameter-specific adaptive regularization. After training, coefficients recomputed across validation mini-batches are averaged to produce parameter-specific survival thresholds. Across autoencoders, RBMs, vision transformers, and LoRA adaptation, the same mechanism produces different sparsification regimes. Matched-sparsity ablations further show that disrupting the learned parameter–signal correspondence reduces performance. These results support gradient-alignment-guided, validation-batch-calibrated pruning pressure as an alternative to relying exclusively on static parameter statistics.

7 Reproducibility Statement

The appendix provides implementation-level training configurations and SPRBM hyperparameters, including optimizer, learning rate, batch size, training duration, random seeds, model architecture, warm-up schedule, gradient-scale calibration, post-training pressure calibration, pruning-threshold parameters, prunable parameter subsets, and deterministic-execution settings. The implementation follows the training and pruning procedure described in Section 3; the final mask is generated from parameter-specific adaptive coefficients averaged across post-training validation mini-batches.

8 Ethics Statement

This work studies neural-network optimization and model sparsification. The experiments do not use human subjects, personally identifiable information, or sensitive personal data. As with other compression techniques, sparsification may reduce deployment resource requirements but does not remove biases present in the underlying models.

9 AI Assistance Disclosure

Large language models were used to assist with manuscript drafting, language editing, formatting, proofreading, literature retrieval and discovery, research ideation, and code generation. The research problem, scientific hypotheses, mathematical formulation, methodological design, experimental design, evaluation decisions, selection and verification of relevant literature, verification of experimental outputs, and scientific interpretation were determined and verified by the author. All AI-assisted text, code, mathematical expressions, literature suggestions, and reported results were independently reviewed and verified by the author, who takes full responsibility for the contents of the submission.

References

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 2015.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. In *International Conference on Learning Representations*, 2018.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, 2019.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, 2020.

P. Sazdar et al. Domain-aware pruning: Sparsity and domain generalization via regularized probabilistic masking. *arXiv preprint*, 2026.

Yixuan Tang and Yi Yang. Gaprune: Gradient-alignment pruning for domain-aware embeddings. *arXiv preprint*, 2025.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020.

Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *International Conference on Learning Representations*, 2021.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023.

Qingru Zhang et al. Platon: Pruning large transformer models with upper confidence bound of weight importance. In *International Conference on Machine Learning*, 2022.

A Additional Experimental Details

A.1 Training Configuration

The benchmark implementation uses the configurations in Table 3. Unless otherwise stated, the AE, RBM, and ViT experiments are evaluated with five seeds $\{42, 123, 456, 789, 2024\}$.

Table 3: Training configurations used by the reported benchmark implementation.

Experiment	Epochs	Batch	Optimizer	LR	Warm-up
AE-MNIST	30	128	SGD	10^{-3}	5
RBM-MNIST	30	128	SGD	10^{-3}	5
ViT-MNIST	20	64	SGD	10^{-3}	5
ViT-CIFAR10	20	64	SGD	10^{-3}	5
DistilGPT2-LoRA	20	1	AdamW	10^{-4}	10

All optimizers use weight decay 10^{-5}. SGD uses momentum 0.9. The ViT and DistilGPT2-LoRA runs use automatic mixed precision when CUDA is available; AE and RBM use full-precision execution.

A.2 SPRBM Hyperparameters

The reported SPRBM implementation uses

$$\Lambda_{\max} = 0.2, \qquad \alpha = 5.0, \qquad \epsilon = 10^{-8}, \tag{23}$$

with adaptive-scale coefficient

$$\eta = 0.1, \tag{24}$$

and affine threshold-mapping constants

$$\tau_{\min} = 0.005, \qquad \tau_{\max} = 0.05. \tag{25}$$

The mean task-gradient magnitude used to calibrate the adaptive scale is estimated once from $M = 20$ batches before training, after which β remains fixed. All random-number-generator states are restored after this pass so that the calibration does not alter the stochastic training stream. After training, adaptive pressure is estimated by averaging coefficients over the first $K = 20$ validation mini-batches at the final trained parameters. The model remains in training mode during this calibration pass, so stochastic layers, when present, remain active; no optimizer update is performed. The validation split therefore contributes to final-mask construction, whereas the test split is used only for evaluation. With $\Lambda_{\max} = 0.2$, the unscaled primary procedure maps these coefficients to an effective threshold range of approximately 0.005 to 0.014; $\tau_{\max} = 0.05$ is a mapping constant, not the realized upper threshold in the primary runs. The global regularization schedule for the AE, RBM, and ViT experiments increases as

$$0.04 \to 0.08 \to 0.12 \to 0.16 \to 0.20 \tag{26}$$

over the first five epochs and then remains at 0.20. For the fixed-regularization baselines, the implementation searches

$$\{10^{-6}, 5{\times}10^{-6}, 10^{-5}, 5{\times}10^{-5}, 10^{-4}, 5{\times}10^{-4}, 10^{-3}\}$$

for L_1, and

$$\{10^{-5}, 5\times10^{-5}, 10^{-4}, 5\times10^{-4}, 10^{-3}, 5\times10^{-3}\}$$

for L_2. WarmStart-SNIP for LoRA uses 100 warm-start steps that are included within the normal training budget rather than added as extra optimizer steps.

A.3 Model Configurations

Autoencoder. The autoencoder is

$$784 \to 256 \to 64 \to 32 \to 64 \to 256 \to 784,$$

with sigmoid activations after every linear layer.

Restricted Boltzmann machine. The RBM contains 784 visible units and 256 hidden units. Its weight matrix is initialized from a zero-mean Gaussian with standard deviation 0.1; visible and hidden biases are initialized to zero. Training uses one-step contrastive divergence (CD-1).

Vision transformers. Both ViTs use four transformer encoder blocks, embedding dimension 64, four attention heads, feed-forward dimension 128, GELU activations, pre-norm transformer layers, and zero dropout. ViT-MNIST uses 7×7 patches on 28×28 grayscale images. ViT-CIFAR10 uses 4×4 patches on 32×32 RGB images.

DistilGPT2-LoRA. The DistilGPT2 backbone is frozen. LoRA uses rank $r = 8$, scaling parameter 16, dropout 0.05, and no trainable bias. The implementation first requests separate query/value projection targets; because DistilGPT2 uses the fused GPT-2-style attention projection, the reported compatible configuration adapts the `c_attn` module. WikiText-2 is tokenized with the DistilGPT2 tokenizer using maximum sequence length 128, truncation, and fixed-length padding. Reported sparsity is computed only over trainable LoRA-A and LoRA-B matrices.

A.4 Prunable Parameter Set

Only trainable two-dimensional parameters are eligible for pruning. For the RBM, only the matrix W is pruned. For DistilGPT2-LoRA, only LoRA-A and LoRA-B matrices are pruned. For the autoencoder and ViTs, trainable two-dimensional weight matrices are pruned while embedding and positional embedding parameters are excluded.

A.5 Deterministic Execution

Python, NumPy, CPU PyTorch, and all CUDA random generators are seeded consistently. cuDNN deterministic execution is enabled, cuDNN benchmarking is disabled, and PyTorch deterministic algorithms are requested with warning-only behavior for operations without deterministic implementations. DataLoader shuffling uses seeded PyTorch generators and zero worker processes in the local Windows/Jupyter configuration.

A.6 Evaluation Protocol

Task quality is measured using the metric appropriate to each architecture:

- reconstruction metrics for AE and RBM experiments;
- classification accuracy for ViT experiments;
- perplexity for DistilGPT2 LoRA adaptation.

Sparsity is computed over the targeted parameter subset. For LoRA experiments, it therefore refers only to trainable adapter parameters, not the frozen DistilGPT2 backbone. Across all experiments, the validation split supplies the $K = 20$ calibration mini-batches used to construct the final SPRBM mask. Test data are not used for pressure calibration or mask construction and are reserved for final evaluation.

B Training-Dynamics Diagnostics

B.1 Evolution of Gradient Agreement

Additional trajectory plots report the evolution of

$$\gamma_i^{(t)} \tag{27}$$

and its aggregate statistics over training. These measurements characterize changes in the optimization regime and should not be interpreted as evidence that the sign or magnitude of γ alone universally predicts final performance. The main mechanistic evidence remains the matched-sparsity perturbation of the signal–parameter correspondence.

B.2 Evolution of Adaptive Regularization

We additionally report the training-time coefficient

$$\lambda_i^{(t)} = \Lambda_t \sigma\left(\alpha \gamma_i^{(t)}\right). \tag{28}$$

These epoch-wise diagnostics characterize the adaptive regularization process but are not accumulated to construct the final mask. The final $\bar{\lambda}_i$ is computed independently after training by averaging coefficients across validation calibration batches. The mask is therefore not generated directly from a single $\gamma_i^{(t)}$, nor from a temporal average of the logged training trajectory.

C Additional Alignment Ablations

The matched-sparsity analysis evaluates several perturbations.

Random signal. Adaptive values are replaced with random assignments while preserving target sparsity.

Constant signal. All parameters receive equivalent signal values.

Shuffled signal. The learned signal distribution is retained but reassigned across parameter locations.

Magnitude. Parameters are selected according to weight magnitude at the same sparsity level. The shuffled condition separates the statistics of the learned signal from its correspondence with specific parameters. Its degradation supports the interpretation that the parameter–signal mapping contains information relevant to sparsification.

D Layer-Wise Sparsity

Additional tables report sparsity for individual parameter groups. For the evaluated vision transformers, feed-forward layers tend to receive greater sparsification than attention projections. For LoRA adaptation, different sparsity patterns emerge between LoRA-A and LoRA-B matrices. These are descriptive observations of the evaluated configurations and are not claimed as universal properties of vision transformers or LoRA.

E Language Adaptation Details

The language-model experiment sparsifies LoRA parameters in DistilGPT2 while leaving the backbone frozen. Thus, reductions in adapter parameters directly affect adapter storage but should not be interpreted as equivalent reductions in total model parameters or total language-model computation. At approximately 53% adapter sparsity, SPRBM-L_1 is compared with magnitude, random, and SNIP pruning. At a less aggressive operating point, SPRBM-L_2 retains 90,781 of the original 147,456 adapter parameters while maintaining perplexity close to the dense-adapter result.

F Computational Cost

The additional computational cost of SPRBM comes from tracking the complexity-gradient interaction and adaptive coefficients throughout training. Because final sparsity is unstructured, deployment speedups depend on sparse execution support. For LoRA adaptation, storage reduction is the most direct consequence of the reported adapter sparsity.

G Interpretation of Parameter Utility

A direct functional definition of parameter utility would measure the change in model behavior after removing a parameter. SPRBM does not explicitly compute this counterfactual quantity for every weight. Instead, it uses an optimization-based proxy: whether reducing parameter complexity agrees with the task descent direction across the post-training validation calibration batches. Accordingly:

> A parameter receives high pruning pressure when complexity reduction remains comparatively compatible with task descent across the terminal-model calibration batches.

This avoids equating local gradient agreement with a stronger causal claim about global functional importance.

H Scope of the Method

SPRBM should be interpreted as a training-time adaptive sparsification framework rather than a fixed pruning rule. Its central design principle is:

$$\begin{aligned}\text{task–complexity gradient agreement} &\rightarrow \text{post-training calibrated pressure} \\ &\rightarrow \text{sparsification decision.}\end{aligned} \tag{29}$$

Future work may explore alternative complexity functions, explicit temporal estimators, structured variants, larger language models, and modern dynamic pruning baselines.

I Additional Discussion of the First-Order Analysis

For a single parameter,

$$\Delta\theta_i = -\rho\left(g_i^{\text{task}} + q_i g_i^{\text{comp}}\right). \tag{30}$$

The task-gradient contribution predicts

$$-\rho(g_i^{\text{task}})^2, \tag{31}$$

while the complexity term contributes

$$-\rho q_i g_i^{\text{task}} g_i^{\text{comp}}. \tag{32}$$

Therefore:

- if $g_i^{\text{task}} g_i^{\text{comp}} > 0$, the complexity update reinforces local task descent;
- if $g_i^{\text{task}} g_i^{\text{comp}} < 0$, it partially opposes task descent;
- if either gradient is very small, the stabilized value in eq. (8) becomes less decisive.

This motivates the adaptive coefficient but does not imply that repeatedly aligned parameters can always be removed without loss.

J Reproducibility Notes

For reproduction, implementations should distinguish:

1. $\gamma_i^{(t)}$: instantaneous parameter-wise signal;
2. $\lambda_i^{(t)}$: adaptive regularization coefficient;
3. $\bar{\lambda}_i$: post-training validation-batch-averaged pressure statistic;
4. τ_i: parameter-specific pruning threshold;
5. m_i: final survival mask.

Conflating these quantities changes the interpretation and behavior of the algorithm.

K Summary of Empirical Evidence

The empirical evidence supports four observations. First, the same adaptive framework yields different sparsity levels across architectures. Second, matched-sparsity perturbations show that learned parameter–signal correspondence matters beyond the number of removed parameters. Third, layer-wise measurements show non-uniform sparsity within architectures. Fourth, the LoRA experiment shows that the same mechanism can reduce a parameter-efficient adaptation space while retaining language-model quality close to the dense-adapter operating point. Together, these findings motivate gradient-alignment-guided, validation-batch-calibrated pruning pressure as an alternative to treating importance as a fixed parameter statistic.

L Optimization-Trajectory Diagnostics

The trajectory-level measurements provide additional context for the architecture-dependent behavior reported in the main paper. These diagnostics are intended to characterize the optimization regimes induced by SPRBM, rather than to establish a universal relationship between the sign of γ and final model quality.

L.1 Gamma Dynamics Across Architectures

The evolution of the parameter-wise agreement signal differs substantially across architectures and between the L_1 and L_2 variants. For AE-MNIST, the mean agreement signal of SPRBM-L_1 becomes progressively more negative during training, reaching approximately -0.0581 at the final epoch. The corresponding value for SPRBM-L_2 reaches approximately -0.0531. In this setting, the increasingly negative trajectory occurs together with stable improvement of the reported validation metric. RBM-MNIST exhibits a substantially different regime. For SPRBM-L_1, the mean agreement signal reaches approximately -0.2935 around epoch 7, representing the strongest negative excursion among the analyzed trajectories. This period coincides with the largest early performance gap relative to the dense model. The

model subsequently recovers a substantial fraction of this gap during later training. The two vision-transformer experiments show another recurring pattern. For SPRBM-L_1, the mean agreement signal moves from a small negative value to a positive value around epochs 6–7. It reaches approximately +0.0043 on ViT-CIFAR10 and +0.0071 on ViT-MNIST near the end of training. In contrast, SPRBM-L_2 remains in a small negative regime throughout training. These observations demonstrate that the same adaptive mechanism does not produce a common agreement trajectory across architectures. In particular, the sign or magnitude of γ should not be interpreted independently of the architecture and optimization state.

L.2 Performance Trajectories

Final metrics hide several temporal behaviors that are visible in the epoch-wise measurements. On AE-MNIST, SPRBM-L_1 remains better than the dense trajectory across all 30 reported epochs. The difference grows during the earlier portion of training, reaches its largest magnitude around the middle of training, and remains present at the final epoch. RBM-MNIST shows a different behavior. SPRBM-L_1 initially moves away from the dense trajectory and reaches its largest gap during the early training phase. It subsequently exhibits a pronounced recovery, substantially reducing the gap by epoch 30. SPRBM-L_2, in contrast, begins closer to the dense model but progressively converges to a weaker final operating point. For ViT-MNIST, SPRBM-L_1 reaches its best mean validation value around epoch 18 and subsequently degrades during the final epochs. A related but smaller late-training effect is visible for ViT-CIFAR10. These trajectories show that final sparsity and final task performance conceal substantially different optimization regimes. They provide diagnostic context for adaptive training but are not inputs to the post-training pressure calibration or final mask.

L.3 Regularization Schedule

For the AE, RBM, and ViT experiments, the global regularization schedule increases during the first five epochs:

$$0.04 \to 0.08 \to 0.12 \to 0.16 \to 0.20, \tag{33}$$

after which it remains at 0.20. Several important changes in γ occur after this schedule has already reached its plateau. For example, the sign transition observed for SPRBM-L_1 in the ViT experiments occurs after the initial increase has completed. Consequently, the later agreement dynamics cannot be attributed solely to continued growth of the global regularization schedule.

M Seed-Level Stability and Uncertainty

The AE, RBM, and ViT experiments use five independent random seeds:

$$\{42,\ 123,\ 456,\ 789,\ 2024\}. \tag{34}$$

The seed-level behavior differs across architectures. For AE-MNIST, the final improvement of SPRBM-L_1 relative to the dense model is observed for all five seeds. The paired 95% confidence interval for the reported final difference is approximately

$$[-0.001118,\ -0.001076], \tag{35}$$

where lower reconstruction error is better. For RBM-MNIST, the direction is also consistent across seeds, but in the opposite direction. The reported paired confidence intervals relative to Dense are approximately

$$[+0.000602,\ +0.000814] \tag{36}$$

for SPRBM-L_1, and

$$[+0.002555, +0.002785] \tag{37}$$

for SPRBM-L_2. The ViT experiments exhibit greater uncertainty. In ViT-CIFAR10, the reported confidence intervals for the differences relative to Dense include zero for both SPRBM variants. ViT-MNIST similarly indicates that the L_2 variant is close to the dense operating point. Because these comparisons use only five seeds, we treat the confidence interval analysis as descriptive evidence of stability rather than as a basis for broad statistical claims. The main conclusions therefore emphasize effect size, consistency across seeds, sparsity, and matched-sparsity comparisons.

N Detailed Layer-Wise Sparsity

The aggregate sparsity values in the main paper conceal substantial variation between parameter groups.

N.1 Autoencoder

For AE-MNIST, the largest encoder layer receives the strongest sparsification. Representative layer-wise values for SPRBM-L_1 include approximately 27.5% sparsity in `encoder.0.weight`, while intermediate encoder/decoder layers exhibit lower sparsity. The corresponding L_1 and L_2 patterns are relatively similar in this architecture, suggesting that the two complexity functions produce comparable final sparsity allocation despite differences in their optimization trajectories.

N.2 Vision Transformers

For the evaluated vision transformers, the strongest sparsification generally appears in the second linear transformation of the feed-forward blocks. For ViT-CIFAR10 under SPRBM-L_1, representative `linear2.weight` sparsities range from approximately 39% to 44%. The corresponding attention output projections receive lower sparsification. The same qualitative pattern is stronger on ViT-MNIST. Table 4 reports representative `linear2.weight` sparsities for this setting.

Table 4: Representative feed-forward layer sparsity for ViT-MNIST with SPRBM-L_1.

Parameter group	Sparsity
`blocks.0.linear2.weight`	$\approx 45.4\%$
`blocks.1.linear2.weight`	$\approx 47.5\%$
`blocks.2.linear2.weight`	$\approx 48.5\%$
`blocks.3.linear2.weight`	$\approx 48.8\%$

Attention-related parameters in the same model are reported at approximately 35% sparsity. These measurements should be interpreted descriptively: they show that the resulting sparsity allocation is non-uniform, but do not establish that feed-forward layers are universally more redundant than attention layers.

O Detailed LoRA Sparsification

The DistilGPT2 experiment targets only the 147,456 trainable LoRA parameters. The frozen backbone is not included in the reported adapter sparsity.

O.1 Matched-Sparsity Baselines

The main paper reports the approximately 53% matched-sparsity comparison including fixed-regularization and SNIP-style alternatives. The appendix also provides the less aggressive approximately 38% operating point and the corresponding detailed values. Table 5 reports the approximately 53% adapter-sparsity comparison.

Table 5: Additional DistilGPT2-LoRA comparison at approximately 53% adapter sparsity. Lower perplexity is better.

Method	Test perplexity
Dense	46.774
SPRBM-L_1	**47.139**
Magnitude	47.397
Fixed-L_2	47.401
Fixed-L_1	47.437
WarmStart-SNIP	49.320
Random	51.128
SNIP-at-init	55.862

Table 6 reports the less aggressive operating point at approximately 38% adapter sparsity.

Table 6: Additional DistilGPT2-LoRA comparison at approximately 38% adapter sparsity.

Method	Test perplexity
Dense	46.774
SPRBM-L_2	**46.846**
Magnitude	46.902
Fixed-L_2	46.917
Fixed-L_1	46.955
WarmStart-SNIP	47.868
Random	49.365
SNIP-at-init	53.926

The approximately 53% comparison shows that SPRBM-L_1 achieves lower perplexity than the tested fixed, magnitude, random, and gradient-snapshot alternatives at the reported matched sparsity. At approximately 38%, SPRBM-L_2 remains close to the dense adapter while outperforming the reported pruning baselines.

O.2 LoRA-A and LoRA-B Allocation

The aggregate adapter sparsity is distributed non-uniformly between the LoRA-A and LoRA-B matrices. For SPRBM-L_1, the reported LoRA-B sparsity across the six transformer blocks ranges approximately from 48.4% to 67.8%, whereas LoRA-A ranges from approximately 22.5% to 40.4%. Table 7 reports representative LoRA-B values across the six transformer blocks.

The L_2 variant is more conservative but retains the same qualitative asymmetry: LoRA-B sparsity is reported between approximately 39.9% and 50.1%, while LoRA-A remains approximately within 13.5%–18.8%. The repeated asymmetry shows that the resulting pruning allocation is not uniform across the two LoRA factors. We treat this as an empirical property of the evaluated DistilGPT2-LoRA configuration rather than as evidence that LoRA-B is universally more redundant than LoRA-A.

Table 7: Representative LoRA-B sparsity across DistilGPT2 transformer blocks under SPRBM-L_1.

Transformer block	LoRA-B sparsity
0	67.8%
1	65.2%
2	62.4%
3	56.9%
4	60.6%
5	48.4%

O.3 Adapter Parameter Counts

The dense LoRA parameter space contains

$$147{,}456 \tag{38}$$

trainable parameters. Table 8 summarizes the two reported SPRBM operating points.

Table 8: Remaining DistilGPT2-LoRA parameters after sparsification.

Method	Sparsity	Remaining	Removed
Dense	0%	147,456	0
SPRBM-L_2	38.44%	90,781	56,675
SPRBM-L_1	52.97%	69,355	78,101

Because the frozen language-model backbone is unchanged, these reductions should be interpreted as adapter compression rather than equivalent reductions in total model size or total inference computation.