BIO-RADDS: BIOLOGICALLY INSPIRED RADIALLY BALANCED DENSE-TO-SPARSE CONVOLUTION FOR EFFICIENT LARGE-KERNEL VISION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-kernel convolutions capture broad spatial context, but their computational cost increases rapidly with kernel area and many spatial connections are essentially redundant. Existing methods either retain dense spatial execution or rely on predefined sparse patterns, limiting their ability to learn efficient spatial connectivity from data. To this end, we present *Bio-inspired Radially Balanced Dense-to-Sparse Convolution (Bio-RadDS)*, a biologically inspired operator that keeps the kernel center dense and learns sparse connections across peripheral radial bands. We combine learnable spatial modulation with a gradual dense-to-sparse transition to obtain compact connectivity while preserving broad spatial coverage. The resulting fixed topology is paired with dedicated sparse execution to enable practical inference acceleration. In the principal operator experiment, Bio-RadDS removes about 60% of the connections in the target kernels and ranks first in 15 of 21 settings across seven image classification datasets. On ConvNeXt-Tiny, the FP32 sparse deployment achieves $2.5\times$–$3.5\times$ end-to-end acceleration as the input size increases from 224 to 2048. These results show that Bio-RadDS learns compact large-kernel connectivity while maintaining strong classification accuracy and enabling efficient deployment at demanding input resolutions.

1 INTRODUCTION

Convolutional neural networks (CNNs) (He et al., 2016; Krizhevsky et al., 2012) have long been a strong foundation for visual recognition. However, their dominance has been challenged by vision Transformers such as ViT (Dosovitskiy, 2020) and Swin (Liu et al., 2021), which offer powerful long-range modeling capabilities. Large-kernel convolution has renewed the spatial modeling ability of CNNs. Architectures such as ConvNeXt (Liu et al., 2022b), RepLKNet (Ding et al., 2022), and UniRepLKNet (Ding et al., 2024) show that convolutional models can remain competitive with ViT by expanding their receptive fields. However, as discussed in LPConv (Kwon et al., 2025), simply enlarging the kernel does not always improve accuracy. In addition to the optimization difficulty, large kernels also introduce substantially higher computational costs. Large-kernel design therefore requires better spatial organization while controlling the increasing computational cost.

Existing methods face a trade-off between preserving spatial expressiveness and reducing the cost of large kernels. SLaK (Liu et al., 2022a) and PeLK (Chen et al., 2024) reduce redundancy by constraining the kernel structure through strip decomposition or peripheral parameter sharing. These constraints improve efficiency but limit the independent modeling of different peripheral positions. RepLKNet (Ding et al., 2022) and LPConv (Kwon et al., 2025) instead preserve the complete support in two dimensions and improve its optimization through structural reparameterization or continuous spatial modulation, as illustrated in Figure 1. However, both methods still evaluate every spatial offset during dense inference. The two directions therefore leave an open question: *whether a large kernel can preserve precise local responses and distributed long-range information while learning a fixed set of active positions that can be executed sparsely.*

Human vision distributes spatial resolution unevenly across the visual field (Wässle et al., 1989). Retinal ganglion cells are densely distributed near the fovea and become progressively sparser toward the periphery (Curcio & Allen, 1990; Wässle et al., 1990). This organization provides high spatial resolution at the center while maintaining broad peripheral coverage. It has inspired foveated

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.

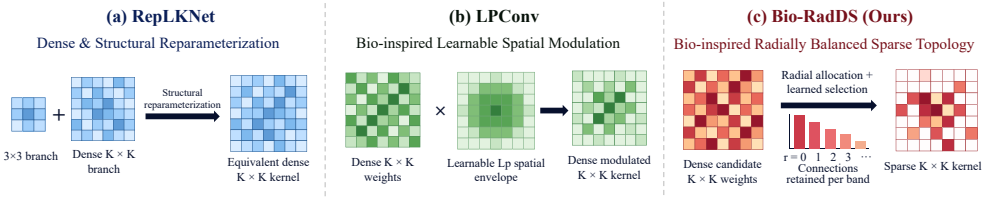


Figure 1: Comparison of spatial support across the illustrated operators. Our method converts dense modulation into a static topology with a protected center and radially distributed connections.

multiresolution systems that preserve fine detail near the viewing center and progressively reduce resolution with eccentricity (Geisler & Perry, 1998). This center–periphery organization suggests a useful principle for large kernels: through a carefully tailored spatial design, spatial connectivity can be reduced while preserving both local detail and broad spatial coverage.

Guided by this principle, we propose **Bio-inspired Radially Balanced Dense-to-Sparse Convolution (Bio-RadDS)** to reduce large-kernel inference cost while maintaining recognition accuracy. We retain a dense center and distribute the remaining connections across radial bands, preventing the selected positions from concentrating near the kernel center. We use a learnable spatial envelope to modulate the dense candidate weights and select peripheral positions according to their effective weight magnitude. We then freeze the selected topology and apply a calibrated cosine schedule to gradually suppress rejected connections and mitigate disruption during sparsification. Finally, we fold the envelope and mask into the retained weights and use static sparse Triton/CUDA kernels to skip inactive offsets, translating the learned connectivity into practical acceleration.

Experiments across adapted conventional backbones and a native large-kernel architecture demonstrate a favorable balance between accuracy and efficiency with substantially fewer active spatial connections. Bio-RadDS obtains the best result in 15 of 21 controlled settings and reaches a mean accuracy across five datasets of 75.49% on RepLKNet-31T, compared with 73.48% for the Dense-LK baseline. With static-sparse deployment, Bio-RadDS achieves $2.5\times$–$3.5\times$ FP32 acceleration on ConvNeXt-Tiny, with larger gains at higher input resolutions.

Our contributions in this work are summarized as follows:

- We propose a biologically inspired large-kernel topology that uses a dense center for local detail and a sparse periphery for broad coverage across spatial scales.
- We formulate effective weight projection under radial constraints, using the selected support as the norm reference during the transition to sparsity.
- We encode each retained spatial index with its input channel weight vector, enabling direct sparse execution with a distinct topology for each output channel.
- Across ten datasets and four backbones, Bio-RadDS maintains high accuracy while removing redundant connections in large kernels and achieving practical acceleration.

2 Related Work

Large-kernel convolution. Conventional CNNs enlarge their receptive fields by stacking small kernels (Simonyan & Zisserman, 2014; He et al., 2016; Huang et al., 2017; Xie et al., 2017). However, their effective receptive fields often occupy only a fraction of the theoretical range (Luo et al., 2016). GCN (Peng et al., 2017) directly expands spatial aggregation through decomposed large kernels. Large depthwise convolution has also become a competitive alternative for modern visual recognition (Liu et al., 2022b; Ding et al., 2022; 2024). To control computational cost, some methods factorize convolution or divide processing among branches (Szegedy et al., 2015; 2016; Yu et al., 2024). Strip kernels combined with dynamic sparsity provide another way to enlarge spatial coverage (Liu et al., 2022a). Other approaches perform filtering in the wavelet domain (Finder et al., 2024) or aggregate features through shifts (Li et al., 2025). Matrix multiplication can also be reorganized to improve execution efficiency (Song et al., 2026). These methods improve performance, but large kernels still need flexible coverage in two dimensions with fewer connections.

Biologically inspired spatial modeling. Human vision preserves fine detail near the fovea while using lower spatial resolution in the periphery (Curcio & Allen, 1990; Rosenholtz, 2016). This

organization motivates foveated imaging (Geisler & Perry, 1998) and provides a prior for neural spatial encoding (Min et al., 2022). The prior also guides local aggregation (Wang et al., 2025) and interactions between the center and surround (Cai et al., 2026). Within large kernels, it informs nonuniform parameter sharing to reduce peripheral redundancy (Chen et al., 2024) and learnable envelopes for continuous weight modulation (Kwon et al., 2025). Existing biologically inspired designs improve different aspects of large-kernel convolution, but balancing spatial modeling with practical computational efficiency remains an open challenge.

Sparse neural networks. Network sparsification seeks to preserve model accuracy with fewer active parameters (LeCun et al., 1989; Han et al., 2015; Frankle & Carbin, 2018). The retained connectivity can also evolve during training through repeated removal and regrowth of connections (Mocanu et al., 2018; Evci et al., 2020; Lasby et al., 2024). However, parameter reduction does not ensure faster execution. Structured pruning addresses this gap by removing entire filters or channels to retain regular computation (Wen et al., 2016; Li et al., 2017; Liu et al., 2017). Local N:M sparsity imposes finer constraints within weight groups to match supported hardware patterns (Zhou et al., 2021; Mishra et al., 2021). Even with such constraints, practical speed depends on kernels that account for workload balance and memory access (Gale et al., 2020; Park et al., 2024; Zhang et al., 2026). For large kernels, an additional challenge is to preserve broad spatial coverage while learning which spatial positions to retain. Our method selects spatial positions once under fixed radial quotas, preserving a dense center and peripheral coverage while producing a static topology.

3 METHOD

Bio-RadDS learns a fixed sparse large-kernel topology that preserves local detail and peripheral coverage. Figure 2 summarizes the pipeline. Section 3.1 distributes connections through radial constraints, Section 3.2 performs a single projection based on trained weights, Section 3.3 stabilizes sparsification through calibration, and Section 3.4 exports the trained topology to static sparse Triton/CUDA kernels for standard and depthwise convolution.

3.1 RADIALLY CONSTRAINED TOPOLOGY

At each output location, a spatial convolution aggregates input features from positions within its kernel window. Because every output channel has its own kernel weights, we define spatial sparsity separately for its corresponding kernel. Let $\mathcal{U}$ denote the index set of all output channels across the target layers, and let $a \in \mathcal{U}$ index the kernel associated with one of these channels. The kernel has a $k_a \times k_a$ spatial grid $\mathcal{K}_a$ and connects d_a input channels. At each position $p \in \mathcal{K}_a$, the vector $\mathbf{w}_{a,p} \in \mathbb{R}^{d_a}$ contains the weights connecting these input channels to output channel a. We associate this vector with a binary mask entry $m_{a,p} \in \{0, 1\}$, which determines whether the spatial position is retained. The mask therefore acts on all input channel connections at that position, while different output channels can learn different spatial patterns.

Controlling only the total number of active positions does not determine how they are distributed within the kernel. To preserve local detail while maintaining coverage at larger distances, we retain a dense center and partition the remaining positions into B disjoint radial bands.

We first define the dense central region. Let $\mathbf{o}_a$ denote the kernel center and let c_a be a user-specified positive odd integer. The protected center is $\mathcal{F}_a = \{p \in \mathcal{K}_a : \|p - \mathbf{o}_a\|_\infty \le (c_a - 1)/2\}$. All positions in $\mathcal{F}_a$ are retained and excluded from subsequent ranking. We divide the remaining positions into B radial bands. For each peripheral position, its radius and normalized radius are

$$\ell_{a,p} = \|p - \mathbf{o}_a\|_2, \qquad z_{a,p} = \frac{\ell_{a,p} - \ell_a^-}{\max(\ell_a^+ - \ell_a^-, \varepsilon_r)}. \tag{1}$$

Here, ℓ_a^- and ℓ_a^+ are the minimum and maximum peripheral radii, and $\varepsilon_r = 1.19 \times 10^{-7}$ is the FP32 denominator floor used for numerical stability. We assign position p to band $b_{a,p} = \min(B - 1, \lfloor B z_{a,p} \rfloor)$, and use $\mathcal{B}_{a,b}$ to denote the positions in band b, where $b = 0, \ldots, B - 1$. All bands span equal intervals of normalized radius. Figure 2(c) illustrates the setting $B = 3$ used in this paper.

With the radial partition in place, we next determine how many spatial positions are retained and how the peripheral positions are distributed across the radial bands. Given a target activity cap ρ_a, we set the total connection cap to $Q_a = \max(|\mathcal{F}_a|, \lfloor \rho_a k_a^2 \rfloor)$, prioritizing center preservation when the requested cap is smaller than the center. Let $Q_{a,b}$ denote the number of retained spatial positions in band b, and let Q'_a denote the total number of retained peripheral positions. The mask of each

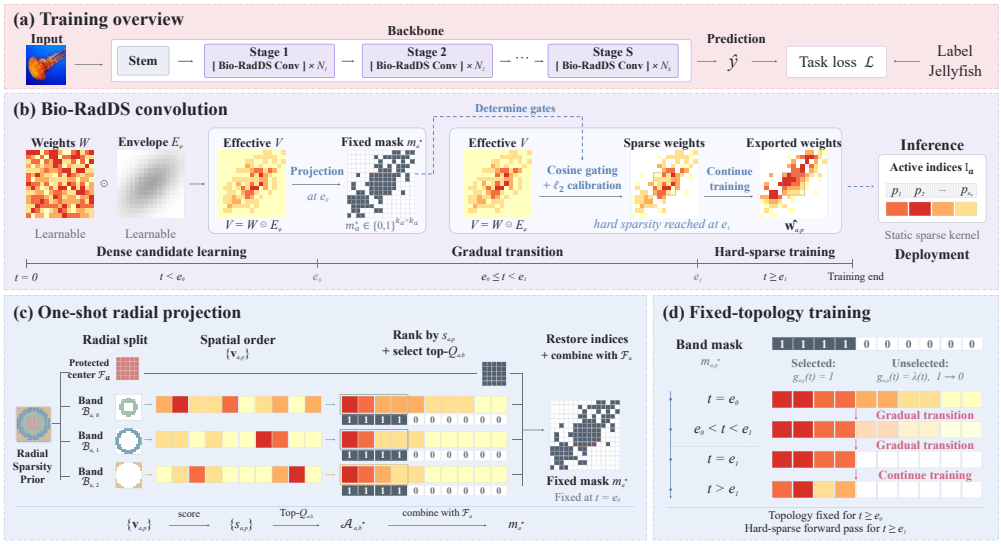


Figure 2: Overview of Bio-RadDS. (a) The backbone is trained with Bio-RadDS operators. (b) A learnable envelope modulates dense weights before sparsification and sparse export. (c) Projection at e_0 preserves the dense center and selects peripheral positions by effective weight magnitude under fixed radial quotas. (d) Unselected gates decay under a cosine schedule with ℓ_2-norm calibration, reaching hard sparsity at e_1, followed by fixed-topology training.

target output kernel must satisfy the following constraints:

$$m_{a,p} = 1 \quad (p \in \mathcal{F}_a), \qquad \sum_{p \in \mathcal{B}_{a,b}} m_{a,p} = Q_{a,b}, \qquad \sum_{b=0}^{B-1} Q_{a,b} = Q'_a \le Q_a - |\mathcal{F}_a|. \tag{2}$$

Equation (2) specifies the required center and band count constraints, while the values of $Q_{a,b}$ remain to be determined. Let $n_{a,b} = |\mathcal{B}_{a,b}|$ denote the number of available positions in band b. To ensure that the number of bands is not reduced due to allocation, each nonempty band first receives κ positions. Subject to the total count constraint in Equation (2) and the available peripheral capacity, we define the total peripheral count Q'_a and the count R_a remaining after these assignments as

$$Q'_a = \min\left(Q_a - |\mathcal{F}_a|, \sum_{b=0}^{B-1} n_{a,b}\right), \qquad R_a = Q'_a - B\kappa. \tag{3}$$

We distribute R_a across the bands in proportion to $r_{a,b} = n_{a,b}\gamma^b$, where γ controls how the allocation changes with radial distance. When $\gamma = 1$, the allocation is proportional to band capacity. For the unsaturated case, this gives the continuous quota

$$x_{a,b} = \kappa + R_a \frac{r_{a,b}}{\sum_{j=0}^{B-1} r_{a,j}}. \tag{4}$$

The values $x_{a,b}$ are generally not integers. We first assign the integer parts of the proportional shares and then distribute the remaining positions in descending order of their fractional remainders. If a band reaches its capacity $n_{a,b}$, it is removed and the unassigned positions are redistributed among the unsaturated bands using the same radial weights. Equal fractional remainders are resolved by a fixed band index order. The resulting integer counts satisfy

$$\sum_{b=0}^{B-1} Q_{a,b} = Q'_a, \qquad \kappa \le Q_{a,b} \le n_{a,b} \quad \text{for all } b. \tag{5}$$

Before training, we verify that every band is nonempty and that Q'_a is feasible under these bounds. Thus, κ is a reserved count for each band, rather than a lower bound imposed after proportional allocation. The realized activity ratio $(|\mathcal{F}_a| + Q'_a)/k_a^2$ can differ from ρ_a because the total cap, band capacities, and integer allocation jointly determine the retained count. All values of $Q_{a,b}$ are fixed before training and remain unchanged during weight optimization.

3.2 DENSE LEARNING FOLLOWED BY A SINGLE PROJECTION

The radial quotas $Q_{a,b}$ determine how many positions each band retains. Training then determines which positions occupy those quotas. Until the projection at e_0, every spatial position remains active so that all candidates can be optimized jointly under the task loss. Following LPConv (Kwon et al., 2025), we use a learnable spatial envelope to modulate the weights of each output kernel:

$$e_{a,p} = \exp\left[-\alpha_a \sum_{j=1}^{2} \left|[\mathbf{C}_a(p - \mathbf{o}_a)]_j\right|^{\nu_a}\right], \qquad \mathbf{v}_{a,p} = e_{a,p}\mathbf{w}_{a,p}. \tag{6}$$

The matrix $\mathbf{C}_a \in \mathbb{R}^{2\times 2}$ controls spatial scale and orientation. The index j refers to the two spatial coordinates. The exponent $\nu_a = 2^{\theta_a}$ determines the contour shape, and $\alpha_a = \mathrm{clip}(\widehat{\alpha}_a, 0, 2)$ controls the modulation strength. We use $\phi = \{\mathbf{C}_a, \theta_a, \widehat{\alpha}_a\}_{a\in\mathcal{U}}$ to denote all learnable envelope parameters and E_ϕ to denote the resulting collection of envelopes. The weights and envelope parameters are optimized jointly, with $\widehat{\alpha}_a$ initialized to one. When $\alpha_a = 0$, the envelope becomes spatially uniform. Equation (6) gives the kernel-level form of $\mathbf{V} = \mathbf{W} \odot E_\phi$ in Figure 2(b).

At e_0, we convert the learned effective weights into a binary mask that satisfies the center and radial quotas defined above. For position p in output kernel a, let $s_{a,p}$ denote its selection score, computed as the mean absolute effective weight across the d_a connected input channels. Let $\mathcal{A}^\star_{a,b}$ denote the set of positions selected from band $\mathcal{B}_{a,b}$. We define the score and selected set as

$$s_{a,p} = \frac{\|\mathbf{v}_{a,p}\|_1}{d_a}, \qquad \mathcal{A}^\star_{a,b} = \mathrm{TopK}\big(\mathcal{B}_{a,b}, Q_{a,b}, (s_{a,p})_{p\in\mathcal{B}_{a,b}}\big). \tag{7}$$

The Top-K operator returns the $Q_{a,b}$ positions in $\mathcal{B}_{a,b}$ with the highest scores. The effective weights are used because they already include the learned envelope used in the deployment representation. Equal scores are resolved by the fixed order of flattened spatial indices. Selection is performed separately for every output kernel and radial band, which satisfies the prescribed quotas without comparing scores across layers or channels.

We map the selected peripheral positions back to the kernel grid and combine them with the protected center. Let $m^\star_{a,p}$ denote the resulting binary mask entry at position p:

$$m^\star_{a,p} = \mathbb{I}\left[p \in \mathcal{F}_a \cup \bigcup_{b=0}^{B-1} \mathcal{A}^\star_{a,b}\right], \qquad m^{(t)}_{a,p} = m^\star_{a,p} \quad (t \geq e_0). \tag{8}$$

Here, $\mathbb{I}[\cdot]$ is the indicator function, and $m^{(t)}_{a,p}$ denotes the binary topology at training time t. Thus, the projected topology is fixed from e_0 onward. Equations (7) and (8) together define the projection illustrated in Figure 2(c). The weights and envelope parameters continue to learn after the topology is fixed. Unselected contributions are not removed immediately. The gates introduced in Section 3.3 gradually suppress them until hard sparsity is reached at e_1.

3.3 TRANSITION AND CALIBRATION WITH A FIXED TOPOLOGY

Section 3.2 describes how projection selects the positions. We now describe how training moves from the dense operator to the final sparse operator while keeping the projected topology fixed. Two specified points in training control this process. Projection is performed at e_0, and hard sparsity is reached at $e_1 > e_0$. The variable t denotes the current training progress, measured on the same scale as e_0 and e_1. Between these two points, the gate $\lambda(t)$ applied to unselected positions follows

$$\lambda(t) = \frac{1}{2}\left[1 + \cos\left(\pi \frac{t - e_0}{e_1 - e_0}\right)\right], \qquad e_0 \leq t < e_1. \tag{9}$$

Outside this interval, we set $\lambda(t) = 1$ for $t < e_0$ and $\lambda(t) = 0$ for $t \geq e_1$. To apply this schedule across the complete kernel, $\bar{m}_{a,p}(t)$ represents the reference mask at position p. It equals one before projection and equals $m^\star_{a,p}$ from e_0 onward. The resulting forward gate $g_{a,p}(t)$ is

$$g_{a,p}(t) = \bar{m}_{a,p}(t) + \lambda(t)\big[1 - \bar{m}_{a,p}(t)\big]. \tag{10}$$

Equation (10) keeps selected positions at a unit gate throughout training. Unselected positions follow $\lambda(t)$ after projection and reach zero at e_1. The projected topology is therefore fixed at e_0, while the forward computation becomes hard-sparse at e_1, as illustrated in Figure 2(d).

Changing the gates also changes the kernel norm even when the effective weights remain fixed. We compensate for this change using the current reference support. For the output kernel indexed by a, $A_a(t)$ is the norm under the reference mask, and $T_a(t)$ is the norm after applying the forward gates.

Both quantities aggregate all spatial positions and connected input channels:

$$A_a(t) = \left(\sum_{p \in \mathcal{K}_a} \|\bar{m}_{a,p}(t)\mathbf{v}_{a,p}\|_2^2 \right)^{1/2}, \qquad T_a(t) = \left(\sum_{p \in \mathcal{K}_a} \|g_{a,p}(t)\mathbf{v}_{a,p}\|_2^2 \right)^{1/2}. \tag{11}$$

We set the numerical threshold to $\varepsilon_{\text{norm}} = 10^{-12}$. The scalar $\beta_a(t)$ rescales the gated weights of the kernel indexed by a to obtain the calibrated vector $\widetilde{\mathbf{w}}_{a,p}(t)$:

$$\widetilde{\mathbf{w}}_{a,p}(t) = \beta_a(t)\, g_{a,p}(t)\mathbf{v}_{a,p}, \qquad \beta_a(t) = \begin{cases} A_a(t)/T_a(t), & T_a(t) > \varepsilon_{\text{norm}}, \\ 1, & T_a(t) \le \varepsilon_{\text{norm}}. \end{cases} \tag{12}$$

When $T_a(t) > \varepsilon_{\text{norm}}$, the calibrated kernel has norm $A_a(t)$. The reference norm is recomputed from the current weights and can therefore change during training. Once hard sparsity is reached at e_1, the forward gate equals the fixed mask, giving $A_a(t) = T_a(t)$ and $\beta_a(t) = 1$. Training then continues on the fixed sparse support, where the retained weights and envelope parameters remain learnable. Appendix A.4 provides the corresponding analysis and nondegeneracy conditions.

3.4 Static Sparse Export and Triton/CUDA Execution

After training on the fixed hard-sparse support, the calibration factor in Equation (12) equals one. We therefore fold the learned envelope and projected mask into the exported weights without changing the learned operator. For the kernel indexed by $a \in \mathcal{U}$, $\widehat{\mathbf{w}}_{a,p}$ is the exported weight vector at position p, $\mathcal{I}_a$ contains the retained spatial positions, and n_a is its cardinality:

$$\widehat{\mathbf{w}}_{a,p} = m^\star_{a,p} e_{a,p} \mathbf{w}_{a,p}, \qquad \mathcal{I}_a = \{p : m^\star_{a,p} = 1\}, \qquad n_a = |\mathcal{I}_a|. \tag{13}$$

The compact representation stores only the weight vectors indexed by $\mathcal{I}_a$, together with their spatial offsets, the bias, and the convolution geometry. Weight packing is completed before inference. For each output location, the Triton/CUDA kernel traverses the retained offsets, reads the corresponding input features, and accumulates their weighted contributions. The spatial loop therefore contains n_a positions instead of all k_a^2 positions, so products and accumulations associated with inactive positions are not executed. The implementation supports both depthwise convolution and standard convolution with $\text{groups} = 1$ through the same retained-offset traversal and the corresponding channel mapping. For the kernel indexed by $a \in \mathcal{U}$, $d_a = C_{\text{in},a}/\text{groups}_a$ is the number of connected input channels, and $H_a \times W_a$ is the output spatial size, where $C_{\text{in},a}$ and groups_a denote the input-channel count and group count of the layer containing the kernel indexed by a. Summing over all target indices gives the dense and sparse MAC counts:

$$M_{\text{dense}} = \sum_{a \in \mathcal{U}} H_a W_a d_a k_a^2, \qquad M_{\text{sparse}} = \sum_{a \in \mathcal{U}} H_a W_a d_a n_a. \tag{14}$$

When the target kernels share spatial size k and active count n, the arithmetic reduction factor is k^2/n. The measured end-to-end acceleration also depends on indexing, memory access, kernel launch costs, and the layers that remain unchanged. We therefore report synchronized inference speedup as $S = T_{\text{dense}}/T_{\text{sparse}}$, with both latency measurements obtained over the complete forward pass using the same hardware, input, precision, and timing protocol.

4 Experiments

We conduct comprehensive evaluations on both conventional backbones and native large-kernel architectures, covering classification performance, sparse deployment, and spatial behavior. In addition, we examine the trade-off between predictive accuracy and computational efficiency to assess the practical effectiveness of our method across different architectural settings.

Datasets and architectures. We use CIFAR-10/100 (Krizhevsky & Hinton, 2009), Tiny-ImageNet (Le et al., 2015), Flowers-102 (Nilsback & Zisserman, 2008), STL-10 (Coates et al., 2011), DTD (Cimpoi et al., 2014), Oxford-IIIT Pets (Parkhi et al., 2012), SVHN (Netzer et al., 2011), Caltech-101 (Fei-Fei et al., 2004), and SUN397 (Xiao et al., 2010). Table 1 evaluates seven datasets, and Table 2 evaluates five. The backbones are ResNet-18/34 (He et al., 2016), ConvNeXt-Tiny (Liu et al., 2022b), and RepLKNet-31T (Ding et al., 2022). They cover adapted conventional networks and a native large-kernel architecture.

Operator comparison across backbones. We examine whether reducing spatial connectivity preserves the accuracy of large-kernel models across architectures. Table 1 compares five operators on three backbones over seven datasets using 224×224 inputs. All variants of each backbone are fine-tuned for 30 epochs from a common adapted dense checkpoint with expanded kernels to en-

sure consistent starting conditions, as detailed in Appendix B.2. Our method retains approximately 40% of the target spatial positions. On the ResNet models with 18 and 34 layers, mean accuracy improves by 1.15 and 1.69 percentage points over the dense baseline, respectively. Our method also ranks first on six of seven datasets for each ResNet, indicating that the higher mean is not driven by a single benchmark. These results show that dense spatial connectivity is not necessary to maintain recognition performance in the evaluated large-kernel models. LPConv remains close to the dense baseline on both ResNets, so continuous spatial modulation alone does not reproduce the gains of the complete sparse procedure in this comparison. For ConvNeXt-Tiny, mean accuracy is essentially unchanged at 87.40% compared with 87.41%. Its benefit is therefore lower connectivity at similar average accuracy. The measured inference latencies further show that reduced connectivity translates into practical acceleration across the evaluated backbones. The speedup varies with the computational share of the target operators and the efficiency of their sparse execution.

Table 1: Top-1 accuracy (%) on seven datasets, with costs for the full network at 224^2. Bold black, blue, and green mark first, second, and third place; blue shading denotes our method.

Backbone	Operator	Top-1 accuracy (%) ↑							Mean	FLOPs	Latency
		CIFAR-100	Flowers-102	DTD	Pets	SVHN	Caltech-101	SUN397	Avg ↑	G ↓	ms ↓
ResNet-18	Dense-LK (2022)	80.64	90.63	63.40	87.22	**97.05**	92.68	54.85	80.92	19.38	8.485
	LPConv (2025)	80.44	90.76	63.19	86.78	96.82	92.86	55.24	80.87	19.38	8.604
	SLaK-style (2022a)	80.29	90.23	63.35	85.83	96.74	91.33	52.90	80.10	27.66	13.543
	PeLK-style (2024)	80.21	90.16	63.03	86.67	96.63	90.62	54.47	80.26	19.38	8.644
	Bio-RadDS (Ours)	**81.88**	**91.84**	**65.74**	**88.44**	96.95	**93.45**	**56.21**	**82.07**	**7.55**	**5.148**
ResNet-34	Dense-LK (2022)	81.00	89.19	63.88	87.68	96.78	92.33	54.99	80.84	39.52	18.336
	LPConv (2025)	81.31	88.79	63.56	86.45	**97.08**	92.80	55.26	80.75	39.52	18.432
	SLaK-style (2022a)	80.40	88.11	62.07	85.66	96.95	91.62	53.42	79.75	56.44	28.325
	PeLK-style (2024)	80.67	89.32	62.45	86.81	96.89	92.57	55.17	80.55	39.52	18.712
	Bio-RadDS (Ours)	**82.57**	**91.67**	**65.32**	**89.18**	97.02	**94.16**	**57.78**	**82.53**	**15.36**	**10.407**
ConvNeXt-Tiny	Dense-LK (2022)	88.41	92.83	73.99	94.47	97.50	96.81	67.84	**87.41**	9.66	11.467
	LPConv (2025)	88.42	92.88	73.88	94.28	97.36	96.93	67.44	87.31	9.66	11.480
	SLaK-style (2022a)	88.42	92.93	**74.04**	94.14	**97.55**	96.99	67.19	87.32	9.34	11.799
	PeLK-style (2024)	**88.55**	92.80	73.24	94.06	97.51	96.81	**68.20**	87.31	9.66	11.083
	Bio-RadDS (Ours)	88.54	**92.97**	73.94	**94.66**	97.41	**97.05**	67.23	87.40	**9.09**	**4.565**

Native large-kernel evaluation. To evaluate the method in a native large-kernel architecture, we train RepLKNet-31T from scratch. All three operator variants are trained from random initialization for 300 epochs using the same optimization protocol and 224×224 inputs. The original kernel sizes are preserved, and our method retains approximately 60% of the target spatial positions. As shown in Table 2, our method achieves the highest accuracy on Tiny-ImageNet, Flowers-102, and STL-10, while remaining close to the strongest baseline on CIFAR-10 and CIFAR-100. Averaged across the five datasets, it reaches 75.49%, exceeding Dense-LK and LPConv by 2.01 and 1.28 percentage points, respectively. These results show that removing approximately 40% of the target connections preserves competitive performance in a backbone designed around native large kernels and supports its cross-architecture applicability.

Table 2: Accuracy (%) of checkpoints from 300-epoch RepLKNet-31T runs.

Dataset	Dense-LK	LPConv	Ours
CIFAR-10	95.85	**95.86**	95.77
CIFAR-100	79.38	**80.17**	79.66
Tiny-ImageNet	63.96	64.36	**64.94**
Flowers-102	47.16	48.67	**54.01**
STL-10	81.03	81.96	**83.05**

Inference scaling with input size. We measure how the compact sparse deployment scales as the input size increases. Images are resized to spatial sizes from 224×224 to 2048×2048, with model parameters fixed throughout the experiment. The same sparse checkpoint is evaluated through a conventional dense implementation with inactive weights set to zero and through the compact sparse CUDA implementation. Measurements use FP32 on an RTX 3090 with batch size one, and latency is averaged over five synchronized timing rounds. Table 3 shows that the conventional dense path has almost the same latency as the dense baseline despite its lower active connection count. Zeroing weights alone therefore does not provide a measurable speed advantage. The increasing speedup is consistent with larger spatial workloads making the cost of indexing and launching sparse kernels less dominant relative to the computation being avoided.

Table 3: FP32 ConvNeXt-Tiny inference on RTX 3090. The two Ours rows share one checkpoint, while dense execution with inactive weights set to zero still executes dense convolution.

Res.	Execution	Full-net active MACs (G) ↓	Target active MACs (G) ↓	Measured inference		
				Latency (ms) ↓	Throughput (img/s) ↑	Speedup (×) ↑
224	Dense-LK	4.832	0.483	11.467 ± 0.168	87.21	1.000
	LPConv	4.832	0.483	11.480 ± 0.182	87.11	0.999
	Ours (dense)	4.543	**0.193**	11.479 ± 0.171	87.11	0.999
	Ours (sparse)	4.543	**0.193**	**4.565 ± 0.166**	**219.08**	**2.512**
512	Dense-LK	25.246	2.521	22.204 ± 0.678	45.04	1.000
	LPConv	25.246	2.521	22.017 ± 0.452	45.42	1.009
	Ours (dense)	23.733	**1.009**	21.874 ± 0.113	45.72	1.015
	Ours (sparse)	23.733	**1.009**	**7.188 ± 0.356**	**139.11**	**3.089**
768	Dense-LK	56.804	5.673	50.357 ± 1.148	19.86	1.000
	LPConv	56.804	5.673	50.144 ± 0.673	19.94	1.004
	Ours (dense)	53.400	**2.269**	50.032 ± 0.210	19.99	1.007
	Ours (sparse)	53.400	**2.269**	**15.335 ± 0.809**	**65.21**	**3.284**
1024	Dense-LK	100.985	10.086	91.093 ± 1.381	10.98	1.000
	LPConv	100.985	10.086	90.854 ± 0.788	11.01	1.003
	Ours (dense)	94.933	**4.034**	90.695 ± 0.231	11.03	1.004
	Ours (sparse)	94.933	**4.034**	**26.146 ± 0.708**	**38.25**	**3.484**
2048	Dense-LK	403.940	40.344	378.767 ± 1.110	2.64	1.000
	LPConv	403.940	40.344	378.654 ± 0.731	2.64	1.000
	Ours (dense)	379.733	**16.138**	378.407 ± 0.400	2.64	1.001
	Ours (sparse)	379.733	**16.138**	**107.331 ± 0.739**	**9.32**	**3.529**

Accuracy and inference efficiency. We fine-tune separate models on CIFAR-100 with target activity caps from 0.3 to 0.7 in steps of 0.1 and measure inference at 224×224 in FP32 on an RTX 3090 with batch size one. Each sparse execution path is compared with the dense execution of the same checkpoint. Table 4 shows that lower retention improves speed on both ResNets, but both become slower than dense execution at $\rho = 0.7$, where arithmetic savings are insufficient to offset sparse execution overhead. ConvNeXt maintains a speedup above $2.3\times$ throughout the tested range. Accuracy does not improve monotonically with retention: the highest recorded values occur at $\rho = 0.3$ for ResNet-34 and ConvNeXt, and at $\rho = 0.4$ for ResNet-18. Retention should therefore be selected based on the accuracy and latency of each backbone.

Table 4: CIFAR-100 Top-1 accuracy and FP32 speedup at 224^2 with batch size 1.

Backbone	Metric	0.3	0.4	0.5	0.6	0.7
ResNet-18	Top-1 (%)	81.77	81.88	80.97	81.36	81.06
	Speedup (×)	2.237	1.609	1.278	1.051	0.901
ResNet-34	Top-1 (%)	82.82	82.57	82.06	82.14	81.51
	Speedup (×)	2.437	1.762	1.391	1.135	0.967
ConvNeXt-T	Top-1 (%)	88.61	88.54	88.32	88.28	88.37
	Speedup (×)	2.458	2.512	2.349	2.416	2.384

Allocation between the center and periphery. On CIFAR-100, we vary the center size c and radial coefficient γ in the first 31×31 operator of RepLKNet-31T, retaining 576 positions per output channel (Appendix B.6). Figure 3 shows ERF differences relative to $c = 1, \gamma = 1$. Changing the center size produces the strongest redistribution of spatial influence, whereas varying γ induces smaller radial shifts. The two parameters act at different levels. Increasing c reserves more connections for the center at the expense of the periphery, while γ adjusts how peripheral connections are distributed across radial bands. These results highlight that equal connection counts do not imply equivalent spatial sensitivity. Center–periphery allocation can reshape the ERF without adding connections, making spatial organization an important consideration beyond the overall retention ratio.

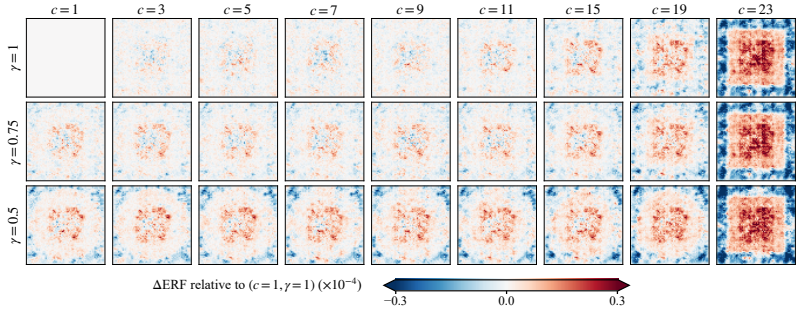


Figure 3: ERF change relative to $c = 1, \gamma = 1$ for the first 31^2 RepLKNet-31T operator. Columns vary center size and rows vary radial allocation while retaining 576 positions.

Dense-to-sparse operator formation. We inspect how the learned spatial representation is converted into a discrete operator for sparse execution. Figure 4 visualizes operators from the first stage of three trained backbones, covering kernel sizes from 7×7 to 31×31. The distinction between the effective weight map and the frozen mask is important. Weight strength describes the learned response, whereas the mask defines the spatial operations required at inference. The resulting index set gives the backend an explicit representation of the learned topology, allowing sparse convolution without reconstructing a dense spatial kernel during execution.

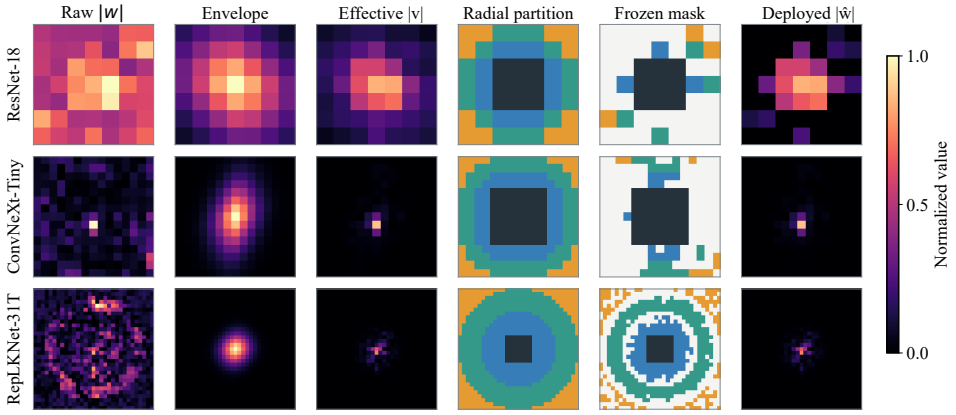


Figure 4: Formation of deployed operators from the first stage in three backbones. Columns trace raw and effective weights, radial partition, frozen mask, and folded weights used by execution.

Topology across stages. Figure 5 examines the first target convolution in each stage of all four backbones. The ResNet models retain broad peripheral connections, deeper ConvNeXt stages place more relative weight near the center, and RepLKNet preserves radial organization across its native kernel sizes. Despite fixed centers and per-band connection counts, the learned arrangements differ across architectures and depths. Thus, the radial prior preserves peripheral coverage without prescribing a single spatial pattern, while training determines the relative emphasis of retained connections.

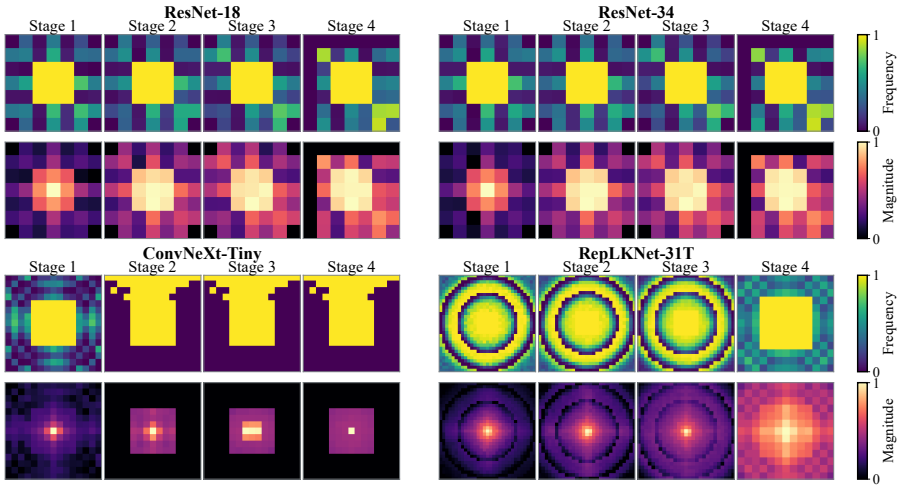


Figure 5: Topology across stages of four backbones. For each stage, the upper row shows mask retention across output channels and the lower row shows normalized deployed weight magnitude.

5 Conclusion

We presented Bio-RadDS for efficient large-kernel convolution. Across diverse backbones, fewer active spatial positions can match or improve accuracy while preserving peripheral coverage. More connections do not consistently enhance accuracy, nor does lower connectivity necessarily reduce latency. Our sparse deployment delivers substantial FP32 acceleration, particularly at higher resolutions. Future work will investigate adaptive allocation across layers and radial bands and evaluate sparse execution on broader hardware and tensor shapes.

Reproducibility Statement

We support the reproducibility of this work by describing the proposed method, experimental protocols, evaluated backbones and datasets, and evaluation metrics in the main paper. The appendix provides further implementation details, including data preprocessing, operator initialization and conversion, optimization settings, hyperparameters, and supplementary analyses. The method specifies radial-cap allocation, a single topology selection, progressive sparsification, and compact sparse export. Evaluation splits and checkpoint-selection rules are stated for the reported comparisons.

AI Use Statement

Generative AI tools were used during manuscript preparation to assist with English-language editing, organization, LaTeX formatting, and drafting or revising descriptions based on author-provided experimental results and instructions. They were not used as independent sources of scientific evidence or to generate the reported experimental measurements. The authors manually reviewed the AI-assisted text, calculations, and formatting against the experimental records and cited sources, made all final methodological and interpretive decisions, and take full responsibility for the content of this work.

References

Xinyong Cai, Changbin Sun, Yong Wang, Hongyu Yang, and Yuankai Wu. Pfgnet: A fully convolutional frequency-guided peripheral gating network for efficient spatiotemporal predictive learning. *arXiv preprint arXiv:2602.20537*, 2026.

Honghao Chen, Xiangxiang Chu, Yongjian Ren, Xin Zhao, and Kaiqi Huang. Pelk: Parameter-efficient large kernel convnets with peripheral convolution. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5557–5567. IEEE, 2024.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3606–3613, 2014.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

Christine A Curcio and Kimberly A Allen. Topography of ganglion cells in human retina. *Journal of comparative Neurology*, 300(1):5–25, 1990.

Xiaohan Ding, X. Zhang, Yizhuang Zhou, Jungong Han, Guiguang Ding, and Jian Sun. Scaling up your kernels to 31×31: Revisiting large kernel design in cnns. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11953–11965, 2022.

Xiaohan Ding, Yiyuan Zhang, Yixiao Ge, Sijie Zhao, Lin Song, Xiangyu Yue, and Ying Shan. Unireplknet: A universal perception large-kernel convnet for audio, video, point cloud, time-series and image recognition. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5513–5524. IEEE, 2024.

Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

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*, pp. 2943–2952. PMLR, 2020.

Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pp. 178–178. IEEE, 2004.

Shahaf E Finder, Roy Amoyal, Eran Treister, and Oren Freifeld. Wavelet convolutions for large receptive fields. In *European conference on computer vision*, pp. 363–380. Springer, 2024.

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

Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14. IEEE, 2020.

Wilson S Geisler and Jeffrey S Perry. Real-time foveated multiresolution system for low-bandwidth video communication. In *Human vision and electronic imaging III*, volume 3299, pp. 294–305. SPIE, 1998.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 2261–2269. Ieee, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Jea Kwon, Sungjun Lim, Kyungwoo Song, and Chulhee Lee. Brain-inspired l_p-convolution benefits large kernels and aligns better with visual cortex. In *International Conference on Learning Representations*, volume 2025, pp. 17854–17884, 2025.

Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. Dynamic sparse training with structured sparsity. In *International Conference on Learning Representations*, volume 2024, pp. 32982–33010, 2024.

Yann Le, Xuan Yang, et al. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

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

Dachong Li, Li Li, Zhuangzhuang Chen, and Jianqiang Li. Shiftwiseconv: Small convolutional kernel with large kernel effect. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 25281–25291. IEEE, 2025.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Tommi Kärkkäinen, Mykola Pechenizkiy, Decebal Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv preprint arXiv:2207.03620*, 2022a.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF international conference on computer vision (ICCV)*, pp. 9992–10002. Ieee, 2021.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *2017 IEEE international conference on computer vision (ICCV)*, pp. 2755–2763. Ieee, 2017.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 11966–11976. IEEE, 2022b.

Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

Juhong Min, Yucheng Zhao, Chong Luo, and Minsu Cho. Peripheral vision transformer. *Advances in Neural Information Processing Systems*, 35:32097–32111, 2022.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 4. Granada, 2011.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*, pp. 722–729. IEEE, 2008.

Cheonjun Park, Mincheol Park, Hyunchan Moon, Myung Kuk Yoon, Seokjin Go, Suhyun Kim, and Won Woo Ro. Deprune: Depth-wise separable convolution pruning for maximizing gpu parallelism. *Advances in Neural Information Processing Systems*, 37:106906–106923, 2024.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pp. 3498–3505. IEEE, 2012.

Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters—improve semantic segmentation by global convolutional network. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 1743–1751. IEEE, 2017.

Ruth Rosenholtz. Capabilities and limitations of peripheral vision. *Annual review of vision science*, 2(1):437–457, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Wan Song, Wei Zhou, Rui Wang, Jun Yu, Toru Kurihara, Jiajia Xu, and Shu Zhan. Wbmm: Windowed batch matrix multiplication for efficient large receptive field convolution. *arXiv preprint arXiv:2607.02097*, 2026.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Ao Wang, Hui Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Lsnet: See large, focus small. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9718–9729. IEEE, 2025.

Heinz Wässle, Ulrike Grünert, Jürgen Röhrenbeck, and Brian B Boycott. Cortical magnification factor and the ganglion cell density of the primate retina. *Nature*, 341(6243):643–646, 1989.

Heinz Wässle, Ulrike Grünert, Jürgen Röhrenbeck, and Brian B Boycott. Retinal ganglion cell density and cortical magnification factor in the primate. *Vision research*, 30(11):1897–1911, 1990.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pp. 3485–3492. IEEE, 2010.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 5987–5995. IEEE, 2017.

Weihao Yu, Pan Zhou, Shuicheng Yan, and Xinchao Wang. Inceptionnext: When inception meets convnext. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5672–5683. IEEE, 2024.

Zhiyi Zhang, Yang Zhao, Jingwei Sun, Liang Qiao, Junshi Chen, Jun Shi, and Hong An. Flash-dwc: Making depthwise convolution compute-efficient on gpus. *ACM Transactions on Architecture and Code Optimization*, 23(4):1–25, 2026.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n: m fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010*, 2021.

APPENDIX CONTENTS

A Properties of the sparse operator . . . 15
A.1 Spatial coverage under radial constraints . . . 15
A.2 Feasible integer allocation . . . 15
A.3 Projection with fixed radial counts . . . 15
A.4 Norm calibration during the transition . . . 16
A.5 Equivalence of dense and sparse evaluation . . . 16
B Implementation and evaluation . . . 16
B.1 Data partitions and preprocessing . . . 17
B.2 Source models and operator conversion . . . 18
B.3 Optimization and sparsity schedules . . . 18
B.4 Checkpoint selection . . . 19
B.5 Execution and measurement . . . 20
B.6 Spatial measurement protocol . . . 21
C Supplementary experiments . . . 22
C.1 Effect of removing radial constraints . . . 22
C.2 Spatial response and peripheral utilization . . . 22
C.3 Individual masks and directional allocation . . . 22
C.4 Variation in spatial utilization across channels . . . 23
C.5 Accuracy under changes in evaluation size . . . 23
C.6 Accuracy and speed at the measured operating points . . . 25
C.7 Numerical agreement after export . . . 25
C.8 Dense CUDA control for sparse acceleration . . . 26
C.9 FP16 inference across input sizes . . . 26
C.10 Variation in latency across layers . . . 27

A PROPERTIES OF THE SPARSE OPERATOR

We fix one output channel and omit its index. Following Section 3, $\mathbf{v}_p \in \mathbb{R}^d$ denotes the effective weight vector at spatial position p. A binary entry m_p selects all d connections in this vector.

A.1 SPATIAL COVERAGE UNDER RADIAL CONSTRAINTS

Let $\mathcal{I}(m) = \{p : m_p = 1\}$ be the retained support. For each nonempty radial band $\mathcal{B}_b$, define the minimum Euclidean distance of its candidate positions from the kernel center o by

$$\ell_b = \min_{p \in \mathcal{B}_b} \|p - o\|_2. \tag{15}$$

Each feasible mask retains $Q_b \geq \kappa \geq 1$ positions in every band. Its maximum connection radius is therefore bounded below by the inner radius of the outermost band:

$$R(m) = \max_{p \in \mathcal{I}(m)} \|p - o\|_2 \geq \max_{0 \leq b < B} \ell_b = \ell_{B-1}. \tag{16}$$

Choose one retained position in each band. Its distance from the center is at least ℓ_b. Taking the maximum over these positions gives Equation (16), independently of their weight ranking.

Let $L(m)$ be the odd side length of the smallest centered square containing the retained support. The relation $\|p - o\|_2 \leq \sqrt{2}\|p - o\|_\infty$ gives the following bound on the integer grid:

$$L(m) = 2 \max_{p \in \mathcal{I}(m)} \|p - o\|_\infty + 1 \geq 2 \left\lceil \frac{\ell_{B-1}}{\sqrt{2}} \right\rceil + 1. \tag{17}$$

The radial constraints therefore impose a minimum spatial extent on every feasible mask. Within that support, the learned weights determine the contribution assigned to each retained position.

A.2 FEASIBLE INTEGER ALLOCATION

Let n_b be the capacity of band b. Assuming $n_b \geq \kappa$ for every band, an integer allocation with peripheral total Q' is feasible if and only if

$$B\kappa \leq Q' \leq \sum_{b=0}^{B-1} n_b. \tag{18}$$

Necessity follows by summing the per-band lower and upper bounds. For sufficiency, initialize $q_b = \kappa$ for every band. The remaining $R = Q' - B\kappa$ positions fit within the unused capacity $\sum_b (n_b - \kappa)$ and can therefore be assigned without violating any upper bound.

The implementation uses capacity-aware largest-remainder allocation. At each pass, let $\mathcal{E} = \{b : q_b < n_b\}$ contain the unsaturated bands. With R positions still unassigned, their proportional shares are

$$s_b = R \frac{n_b \gamma^b}{\sum_{j \in \mathcal{E}} n_j \gamma^j}, \qquad b \in \mathcal{E}. \tag{19}$$

The integer part of each s_b is added subject to the residual capacity $n_b - q_b$. Remaining positions are then assigned in descending order of the fractional parts of s_b, with ties resolved by band index. If capacity clipping leaves positions unassigned, the procedure repeats over the unsaturated bands. For the 15×15 kernel with a 7×7 center, $(n_0, n_1, n_2) = (60, 84, 32)$, $Q' = 41$, and $\gamma = \kappa = 1$. After reserving $(1, 1, 1)$, the remaining 38 positions have shares $(12.9545, 18.1364, 6.9091)$, giving the final allocation $(14, 19, 8)$ used by the reported masks. Invalid configurations are rejected before training.

A.3 PROJECTION WITH FIXED RADIAL COUNTS

At projection, the effective weights and radial counts are fixed. Let $\mathcal{M}$ contain the masks satisfying the prescribed center and band counts. For $m \in \mathcal{M}$, define the removed absolute weight mass by

$$E(m) = \sum_{p \in \mathcal{K}} (1 - m_p) \|\mathbf{v}_p\|_1. \tag{20}$$

Within one output channel, the scoring divisor d is constant. Ranking by $\|\mathbf{v}_p\|_1 / d$ is therefore equivalent to ranking by $\|\mathbf{v}_p\|_1$. Selecting the Q_b highest scores in each band yields

$$m^\star \in \arg\min_{m \in \mathcal{M}} E(m). \tag{21}$$

Suppose a feasible mask retains a smaller score and omits a larger score in the same band. Exchanging these positions preserves the constraints and cannot increase E. Repeating this exchange produces the selected mask. Since the bands are disjoint, the selection is optimal within $\mathcal{M}$.

For one output location, let $\mathbf{x}_p \in \mathbb{R}^d$ contain the connected input values at position p. Hold the weights and bias fixed. Suppose each input component varies independently in $[-M, M]$ and all relevant positions lie inside the input domain. The maximum output deviation then satisfies

$$\sup_{\|\mathbf{x}\|_\infty \le M} \left| \sum_{p \in \mathcal{K}} (1 - m_p) \langle \mathbf{v}_p, \mathbf{x}_p \rangle \right| = ME(m). \tag{22}$$

The triangle inequality gives the upper bound. Equality is attained when each removed input component has magnitude M and the sign of its corresponding weight. Consequently, the selected mask minimizes this maximum deviation over the same feasible family $\mathcal{M}$.

A.4 Norm calibration during the transition

Fix the current effective weights and selected mask. Let A and D denote the norms of the retained and removed parts respectively, aggregated over their spatial positions and connected input channels:

$$A^2 = \sum_{p \in \mathcal{K}} m_p^\star \|\mathbf{v}_p\|_2^2, \qquad D^2 = \sum_{p \in \mathcal{K}} (1 - m_p^\star) \|\mathbf{v}_p\|_2^2. \tag{23}$$

The two parts occupy disjoint positions. Selected weights retain unit gates, while removed weights are multiplied by λ. Their contributions to the squared gated norm therefore satisfy

$$T^2 = A^2 + \lambda^2 D^2. \tag{24}$$

When $T > \varepsilon_{\text{norm}}$, calibration with $\beta = A/T$ gives a kernel norm equal to A. The reference norm is recomputed from the current weights at every forward pass and can change during optimization. The two norms and β are computed from detached effective weights inside a no-gradient block, so the calibration factor is treated as a constant during back-propagation. We use $\varepsilon_{\text{norm}} = 10^{-12}$; when $T \le \varepsilon_{\text{norm}}$, the implementation sets $\beta = 1$, and any non-finite ratio is also replaced by one.

At projection time e_0, the reference changes from the dense support to the selected support while $\lambda = 1$. For fixed weights with $\sqrt{A^2 + D^2} > \varepsilon_{\text{norm}}$, the resulting calibration factor is

$$\beta(e_0) = \frac{A}{\sqrt{A^2 + D^2}}. \tag{25}$$

The reference change can rescale the effective kernel at projection. Once $\lambda = 0$, the gate equals the selected mask and $T = A$. Calibration then equals one in both implementation branches.

A.5 Equivalence of dense and sparse evaluation

At hard sparsity, the folded weights satisfy $\widehat{\mathbf{w}}_p = m_p^\star \mathbf{v}_p$. Let $\mathbf{x}_p(u)$ be the input vector connected to output coordinate u. With identical geometry and bias, the dense and compact responses satisfy

$$y(u) = b + \sum_{p \in \mathcal{K}} \langle \widehat{\mathbf{w}}_p, \mathbf{x}_p(u) \rangle = b + \sum_{p \in \mathcal{I}(m^\star)} \langle \widehat{\mathbf{w}}_p, \mathbf{x}_p(u) \rangle. \tag{26}$$

Every omitted term is zero, so the restricted sum preserves the output in real arithmetic. Appendix C.7 measures agreement between the implementations under floating-point accumulation.

B Implementation and evaluation

B.1 Data partitions and preprocessing

Table 5 summarizes the data partitions for Tables 1 and 2. Within each study, all methods share the same sample lists. For Table 1, CIFAR-100, Pets, and SVHN reserve stratified validation subsets from their training pools. Flowers-102 and DTD use official validation sets, Caltech-101 uses a fixed stratified partition, and SUN397 uses partition 01 with five training images per class reserved for validation. The partition seed is zero.

For Table 2, CIFAR-10/100 and STL-10 reserve a stratified 10% validation holdout from the official training split with partition seed 20260920; their official test sets are used only after checkpoint and weight selection. Flowers-102 uses its official validation split for selection and its 6,149-image official test split for reporting. Tiny-ImageNet reserves a stratified 10% holdout from the official training split for checkpoint selection and reports the selected checkpoint on the official validation set because the public test labels are unavailable. Checkpoint-selection details are given in Appendix B.4.

Table 5: Data partitions for Tables 1 and 2. Counts are images. The validation column is used for checkpoint selection and the report column for final accuracy.

Dataset	Classes	Train	Validation	Report	Reporting set
Table 1: conventional-backbone comparison					
CIFAR-100	100	45,000	5,000	10,000	Held-out
Flowers-102	102	1,020	1,020	6,149	Held-out
DTD	47	1,880	1,880	1,880	Held-out
Oxford-IIIT Pets	37	3,312	368	3,669	Held-out
SVHN	10	65,937	7,320	26,032	Held-out
Caltech-101	101	6,330	652	1,695	Held-out
SUN397	397	17,865	1,985	19,850	Held-out
Table 2: RepLKNet-31T comparison					
CIFAR-10	10	45,000	5,000	10,000	Official test
CIFAR-100	100	45,000	5,000	10,000	Official test
Tiny-ImageNet	200	90,000	10,000	10,000	Official validation
Flowers-102	102	1,020	1,020	6,149	Official test
STL-10	10	4,500	500	8,000	Official test

Transforms and normalization. For Table 1, training uses a 224×224 random resized crop with scale $(0.08, 1.0)$, aspect ratio $(3/4, 4/3)$, bilinear interpolation, and horizontal flipping with probability 0.5. Evaluation resizes the shorter side to 256 pixels with bilinear interpolation and then applies a 224×224 center crop. All conventional-backbone runs use ImageNet normalization, with mean $(0.485, 0.456, 0.406)$ and standard deviation $(0.229, 0.224, 0.225)$; Mixup, CutMix, RandAugment, and random erasing are disabled.

For Table 2, training uses the same crop scale, aspect-ratio range, and flip probability, followed by the default TorchVision 0.14.1 RandAugment configuration with two operations, magnitude 9, and 31 magnitude bins. Evaluation uses the 256-pixel resize and 224×224 center crop. Random erasing is disabled. CIFAR-10 uses mean $(0.4914, 0.4822, 0.4465)$ and standard deviation $(0.2470, 0.2435, 0.2616)$. CIFAR-100 uses mean $(0.5071, 0.4867, 0.4408)$ and standard deviation $(0.2675, 0.2565, 0.2761)$. STL-10 uses mean $(0.4467, 0.4398, 0.4066)$ and standard deviation $(0.2603, 0.2566, 0.2713)$. Tiny-ImageNet and Flowers-102 use the ImageNet values above. Mixup and CutMix are specified in Appendix B.3.

Sample-list construction. For each CIFAR-10/100 and STL-10 class, a single Python random.Random(20260920) stream shuffles the official training indices; the first $\mathrm{round}(0.1n_c)$ indices form validation and the remainder form training, after which both lists are sorted. The count is clamped to leave at least one sample in each subset. Split fingerprints hash the source split, fraction, seed, role, and exact indices. Flowers-102 uses the official split lists. Tiny-ImageNet reads the official validation annotation file in file order and reuses the class-to-index mapping from the sorted training directories. SVHN excludes the extra set, and STL-10 excludes unlabeled images.

B.2 SOURCE MODELS AND OPERATOR CONVERSION

Source construction. Starting from public TorchVision ImageNet-1K V1 weights, we expand each eligible spatial convolution with an odd square kernel of side length k to

$$K = k + 2\left\lceil \frac{k}{2} \right\rceil. \tag{27}$$

The original weights occupy the center of the expanded kernel. New positions start at zero before five epochs of adaptation on ImageNet-1K. Each operator variant is constructed from the resulting dense checkpoint. Unmodified tensors are copied from that source. The target classifier uses the same initialization seed across methods, and target fine tuning starts with fresh training state.

Table 6: Target layers used in the operator comparisons. Conventional backbones expand eligible kernels while preserving the original region as the dense center. RepLKNet retains its native sizes with a 7×7 center.

Backbone	Layers	Target operator	Kernel treatment
ResNet-18	17	Spatial convolution	Expanded using Equation (27)
ResNet-34	33	Spatial convolution	Expanded using Equation (27)
ConvNeXt-Tiny	18	Depthwise convolution	Expanded using Equation (27)
RepLKNet-31T	12	Large-kernel branch	Retain native sizes

RepLKNet-31T specification. Our RepLKNet-31T is the compact four-stage variant rather than RepLKNet-31B. Its stage depths are $[2, 2, 6, 2]$, its channel widths are $[64, 128, 256, 512]$, and its native depthwise large-kernel side lengths are $[31, 29, 27, 13]$. The four-convolution stem and three downsampling transitions follow the RepLKNet design. Each large-kernel block has a parallel 5×5 branch during training and is followed by a ConvFFN with fourfold hidden width. The backbone contains $11{,}546{,}304$ trainable parameters before the dataset-specific classifier; a 512-to-C linear head adds $513C$ parameters, giving $11{,}597{,}604$ parameters for CIFAR-100. We convert every native large-kernel depthwise branch: using one-based block numbering, stages 1–4 contribute blocks $[1, 2]$, $[1, 2]$, $[1, \ldots, 6]$, and $[1, 2]$, respectively. These $2 + 2 + 6 + 2 = 12$ branches are the complete target-layer list; no stage depth or width is changed. Conversion preserves channel connectivity and stride, and padding is adjusted to retain the output shape. RepLKNet keeps the native kernel sizes and uses a 7×7 protected center.

Envelope initialization. Dense-LK retains the adapted source weights directly. LPConv initializes $C^{(0)} = I_2/(100 + 10^{-4})$ with $\theta^{(0)} = 1$. Bio-RadDS uses the same values and sets $\widehat{\alpha}^{(0)} = 1$. For source weight vector $\mathbf{w}_p^{\mathrm{src}}$, compensation for the initial envelope sets

$$\mathbf{w}_p^{(0)} = \frac{\mathbf{w}_p^{\mathrm{src}}}{e_p^{(0)}}. \tag{28}$$

Envelope compensation makes the effective kernel equal to the source kernel. Its parameters remain trainable, with an initially isotropic coordinate transform. Existing normalization tensors and running statistics are copied from the source and continue updating during target fine tuning.

Adapted structured operators. The SLaK adaptation projects the source kernel onto a $K \times 5$ strip and a $5 \times K$ strip. Overlapping weights and the bias are divided equally between the branches. Positions outside the cross are removed. The PeLK adaptation keeps five central coordinates as separate cells and uses wider sharing cells toward the periphery. Each compact parameter starts at the mean source weight in its cell. Its position embedding starts at zero.

B.3 OPTIMIZATION AND SPARSITY SCHEDULES

Conventional backbones. Source adaptation uses AdamW for five epochs with weight decay 0.01. Two GPUs each process a batch of 32. The initial learning rate is 10^{-4} for both ResNet models and 5×10^{-5} for ConvNeXt-Tiny. One epoch of linear warmup precedes cosine decay. Target fine tuning lasts 30 epochs and uses the same backbone-specific initial learning rates: 10^{-4} for ResNet-18/34 and 5×10^{-5} for ConvNeXt-Tiny. AdamW uses weight decay 0.01, a batch size of 32 per GPU, one warmup epoch, and cosine decay, with validation after every epoch. Training uses mixed precision with dynamic loss scaling and clips the global gradient norm to 1.0. Bio-RadDS envelope parameters C, θ, and $\widehat{\alpha}$ belong to the same AdamW parameter group as the remaining

model parameters; no separate optimizer or learning-rate multiplier is used, and they receive the same weight decay.

Native large-kernel training. RepLKNet-31T is trained from random initialization for 300 epochs. AdamW starts at learning rate 5×10^{-4} and decays to 10^{-5}. Five warmup epochs precede cosine decay. Using zero-based epoch indices, weight decay is 0.05 for epochs 0–109 and 0.01 for epochs 110–299. The main stage uses Mixup with parameter 0.8 and CutMix with parameter 1.0. RandAugment and repeated augmentation are also enabled. At the start of epoch 110, Mixup and CutMix are disabled and label smoothing changes from zero to 0.1. Training uses FP16 mixed precision with EMA decay 0.99.

Table 7: Sparsity schedules. The first row uses fractions of successful optimizer updates. Other times are epochs.

Study	Dense center	Retention	e_0	e_1
Conventional transfer	Original kernel	$\rho = 0.4$	10%	30%
Native architecture	7×7	$\rho = 0.6$	20	40
Spatial allocation	Varied	$n = 576$	3	9

Mask implementation. The principal classification studies use three bands with $\gamma = \kappa = 1$. Each band first receives one reserved position, and the remaining peripheral count is distributed by the capacity-aware largest-remainder procedure in Section 3.1. Projection scores are computed in FP32 from the current effective weights, namely the raw weights multiplied by the learned envelope. Equal scores are resolved by flattened spatial index. One process selects the mask and broadcasts it to the other workers. The binary mask is copied unchanged into the EMA model and restored on resumption. Training progress is stored as the successful optimizer update count. After e_1, inactive positions have zero forward contribution while their raw values remain in the dense training tensors.

B.4 Checkpoint selection

Table 1 selects ordinary model weights using the highest validation accuracy in the shared eligible interval. It uses no EMA model. Bio-RadDS checkpoints become eligible at e_1. The reporting set is evaluated after selection. Each classification setting uses one training seed.

RepLKNet-31T runs. Table 2 uses 300-epoch RepLKNet-31T runs with training seed zero and the data partitions in Table 5.

Checkpoint and weight selection. At each epoch, ordinary and EMA weights are evaluated on the validation set. The higher Top-1 score determines the weight source, with ordinary weights preferred for an exact tie. The highest score across epochs selects the checkpoint, retaining the earlier epoch on a tie. For CIFAR-10/100, Flowers-102, and STL-10, the fixed weight source is reported on the official test set. For Tiny-ImageNet, selection uses the 10% holdout from the official training split, and the fixed checkpoint is reported on the official validation set because public test labels are unavailable. Table 8 lists the selected epoch and weight source for each run.

Table 8: Selected checkpoints for Table 2. Epochs are zero-based; Model and EMA denote ordinary and exponential moving-average weights.

	Dense-LK		LPConv		Bio-RadDS	
Dataset	Epoch	Weights	Epoch	Weights	Epoch	Weights
CIFAR-10	299	EMA	227	EMA	177	EMA
CIFAR-100	179	EMA	289	Model	236	EMA
Tiny-ImageNet	125	EMA	124	EMA	125	EMA
Flowers-102	279	Model	286	Model	283	Model
STL-10	237	Model	288	Model	259	Model

Sparse state at selection. For these runs, projection occurs at $e_0 = 20$ and hard sparsity begins at $e_1 = 40$. Checkpoint selection searches all training epochs rather than explicitly restricting eligibility to $t \geq e_1$. All five selected Bio-RadDS checkpoints are nevertheless after e_1, and their saved states confirm hard sparsity in all twelve target layers. Training and classification evaluation

use FP16; the effective training batch size is 256 for CIFAR-10/100 and STL-10 and 512 for Tiny-ImageNet and Flowers-102, shared across methods within each dataset.

B.5 Execution and measurement

Execution paths. The reference path evaluates a dense convolution with inactive weights set to zero. The compact path uses the same folded checkpoint and visits only retained spatial indices. Other network parameters are identical. The sparse backend supports standard convolution with one group and depthwise convolution. Other grouped layouts use the dense implementation.

Compact representation and indexing. For a layer with Q retained positions per output channel, flattened kernel indices are stored as a contiguous int32 tensor of shape $[C_{\text{out}}, Q]$. Folded weights are stored output-channel-major with shape $[C_{\text{out}}, C_{\text{in}}, Q]$ for standard convolution and $[C, 1, Q]$ for depthwise convolution. Inputs and outputs use contiguous NCHW layout. A flattened index p is decoded as $k_y = \lfloor p/K \rfloor$ and $k_x = p - k_y K$; the corresponding input coordinate is formed from the output coordinate, stride, dilation, and padding, and out-of-range loads are masked to zero. The matched dense Triton control uses the same representation and kernel but sets $Q = K^2$, thereby visiting every spatial offset.

Triton launch and tiling. One Triton program produces a tile of output spatial positions and output channels for one batch element. The depthwise path uses $B_C = 4$ channels and $B_{HW} = 128$ output positions. The standard $\text{groups} = 1$ path reduces input channels in chunks and uses $(B_{C_{\text{out}}}, B_{HW}, B_{C_{\text{in}}}) = (8, 64, 4)$ when $C_{\text{in}} \leq 4$, $(2, 128, 64)$ when $H_{\text{out}} W_{\text{out}} \geq 56^2$ and $C_{\text{in}} \leq 64$, $(8, 32, 32)$ when $H_{\text{out}} W_{\text{out}} \geq 14^2$ and $C_{\text{in}} \leq 256$, and $(4, 64, 64)$ otherwise. These deterministic rules are fixed before timing. The reported static sparse path uses no Triton autotuning and does not override num_warps or num_stages; the compiler defaults of the recorded Triton version are used.

Sparse-kernel pseudocode. For each tile $(n, \mathcal{C}_o, \mathcal{T}_{HW})$ in parallel: (1) initialize an FP32 accumulator; (2) for each retained offset q, load its int32 index, decode (k_y, k_x), load valid NCHW input values, and multiply by the compact weight; (3) for standard convolution, reduce C_{in} in $B_{C_{\text{in}}}$-sized chunks; (4) add bias and store the tile in the requested output dtype. FP16 inputs and weights therefore accumulate in FP32 before the final FP16 store; FP32 execution also uses the FP32 accumulator.

Implementation correspondence. In the accompanying code, ops/envelope.py, ops/projection.py, and ops/conv.py implement the envelope, Equations (7)–(8), radial/global projection, transition, and calibration used to produce the masks summarized in Figure 6. The export and compact layout are implemented in deploy/static_backbone.py; the two Triton kernels are in deploy/triton_sparse.py. The scripts benchmark_same_backend_inline.py and deploy/efficiency_resolution.py implement the synchronized full-forward, inline layer, matched-backend, and input-size protocols reported in Tables 10, 11, and 12.

Hardware and timing. Measurements use an RTX 3090 with Python 3.9.25, PyTorch 1.13.1 built with CUDA 11.7, and Triton 2.1.0. Inputs use NCHW layout with batch size one. FP32 measurements disable TF32 for matrix multiplication and cuDNN. Models run in evaluation and inference modes with inputs already on the GPU. Preprocessing and compilation finish before timing, as does sparse export. The main input size study uses five interleaved rounds. Each contains 50 warmup passes and 200 synchronized forward passes. Latency reports the mean and standard deviation across rounds.

Computation accounting. Target MACs use the realized mask support, and full network counts include unchanged operators. A multiply and accumulate counts as one MAC or two FLOPs. Full model latency is measured directly from the input tensor through the classifier.

Memory accounting. All reported deployment-memory values are CUDA peak allocated memory, not reserved memory. For each deployed model we call reset_peak_memory_stats, allocate one input batch of size 1 on the GPU, run one complete inference_mode forward pass, synchronize, and record max_memory_allocated; MiB denotes bytes divided by 2^{20}. No baseline is subtracted. The peak therefore includes the resident deployed parameters and buffers, the input, activations, and

any allocated operator workspace, but excludes CUDA reserved-yet-unallocated cache. Sparse rows hold the compact folded weights and int32 indices and do not retain a second dense training model.

Input size tests. The deployment sweep resizes the evaluation images to different input sizes while keeping the model fixed. It measures inference latency as the spatial workload increases. The DTD accuracy sweep uses the same images at every size with deterministic resizing and center cropping.

B.6 Spatial measurement protocol

Allocation at a fixed connection count. The spatial study varies the center side c over nine values from 1 to 23 and uses three values of γ. Only the first 31×31 RepLKNet operator changes. Each configuration retains exactly 576 positions per output channel. Let $R = 576 - |\mathcal{F}|$ be the peripheral count and $n_b = |\mathcal{B}_b|$ the capacity of band b. Starting from κ positions in each band, the remaining count is distributed in proportion to $n_b\gamma^b$. Saturated bands stop receiving positions, and their excess is redistributed before deterministic rounding by fractional remainders.

A scalar $\tau \geq 0$ controls the allocation to bands with remaining capacity. It is selected so that the continuous counts after capacity clipping sum to the required peripheral total R:

$$\widetilde{q}_b = \kappa + \min\{n_b - \kappa, \tau n_b \gamma^b\}, \qquad \sum_b \widetilde{q}_b = R. \tag{29}$$

Rounding the continuous counts by their fractional remainders preserves the required total. The resulting integer counts satisfy both the capacity of every band and the fixed support size:

$$|\mathcal{F}| + \sum_b q_b = 576, \qquad \kappa \leq q_b \leq n_b. \tag{30}$$

For $c = 1$ and $\gamma = 0.5$, the capacities $(184, 480, 296)$ produce the allocation $(184, 298, 93)$. Including the central position gives a total of 576. This procedure fixes the count for the spatial study, while the principal classification experiments use the allocation in Section 3.1.

ERF computation. Each configuration is fine-tuned for 30 epochs from the same selected dense checkpoint, with $e_0 = 3$ and $e_1 = 9$. ERF uses the final EMA at zero-based epoch 29 and the output of the first target operator, stages.0.blocks.0.large_kernel.lkb_origin.conv. For an output tensor z, the differentiated scalar is the mean across all channels of $\mathrm{ReLU}(z_{:,:,H/2,W/2})$; no single channel is selected. Gradients are taken with respect to the pre-normalization RGB image, their absolute values are summed over the three input channels, and each per-image map is normalized to unit total mass before averaging. The 100 images are the first official CIFAR-100 test example of each class in dataset order, one per class; all 100 produced finite nonzero maps. Inputs are resized to 256 pixels and center-cropped to 224×224 using the CIFAR-100 normalization above. Difference maps subtract the reference $c = 1, \gamma = 1$, and radial curves average the resulting ERF values within integer-radius bins.

Topology statistics. Retention frequency averages binary masks across output channels. Deployed magnitude averages absolute folded weights across connected channel pairs and is normalized within each layer. Angular plots group retained peripheral positions into eight sectors. For energy statistics, let $\widehat{\mathbf{w}}_{a,p} = m^\star_{a,p}\mathbf{v}_{a,p}$ denote the deployed weight vector at position p for output channel a, with the envelope folded in and inactive positions set to zero. We use squared-weight energy $E_{a,p} = \|\widehat{\mathbf{w}}_{a,p}\|_2^2$, summed over the connected input channels. For a kernel grid $\mathcal{K}$ centered at $\mathbf{o}$, the normalized spatial radius and its energy-weighted channel mean are

$$r_p = \frac{\|p - \mathbf{o}\|_2}{\max\{1, \max_{q \in \mathcal{K}} \|q - \mathbf{o}\|_2\}}, \qquad \bar{r}_a = \frac{\sum_{p \in \mathcal{K}} E_{a,p} r_p}{\max\{\sum_{p \in \mathcal{K}} E_{a,p}, 10^{-30}\}}.$$

For the plotted kernels, the radius normalizer is the center-to-corner distance of the full kernel grid, not the maximum retained radius. Sums include the protected center and all peripheral positions; an all-zero channel has $\bar{r}_a = 0$. Energy composition uses the same squared weights and is normalized by the total deployed energy in each layer. Figure 11 plots $\bar{r}_a$ for all output channels of the last target convolution in each stage. Boxes report the median and interquartile range, with whiskers spanning all channels. Individual masks show output channel zero of those layers. The main stage averages use the first target layer.

C SUPPLEMENTARY EXPERIMENTS

C.1 EFFECT OF REMOVING RADIAL CONSTRAINTS

We compare radial Top-K selection with a global Top-K control using the same effective weights, dense center, and total connection count. Figure 6 shows representative masks and the distribution of mean retained peripheral radius across output channels. Radial selection preserves the outermost band in every displayed ResNet channel, whereas global selection leaves it empty in 5 of 64 ResNet-18 channels and 4 of 64 ResNet-34 channels. The representative global masks also contract to smaller support boxes. ConvNeXt-Tiny has no empty outermost bands under either selector, although its global selection produces a broader radius distribution. These results show that unconstrained global ranking can reduce peripheral coverage even when the number of retained positions is unchanged.

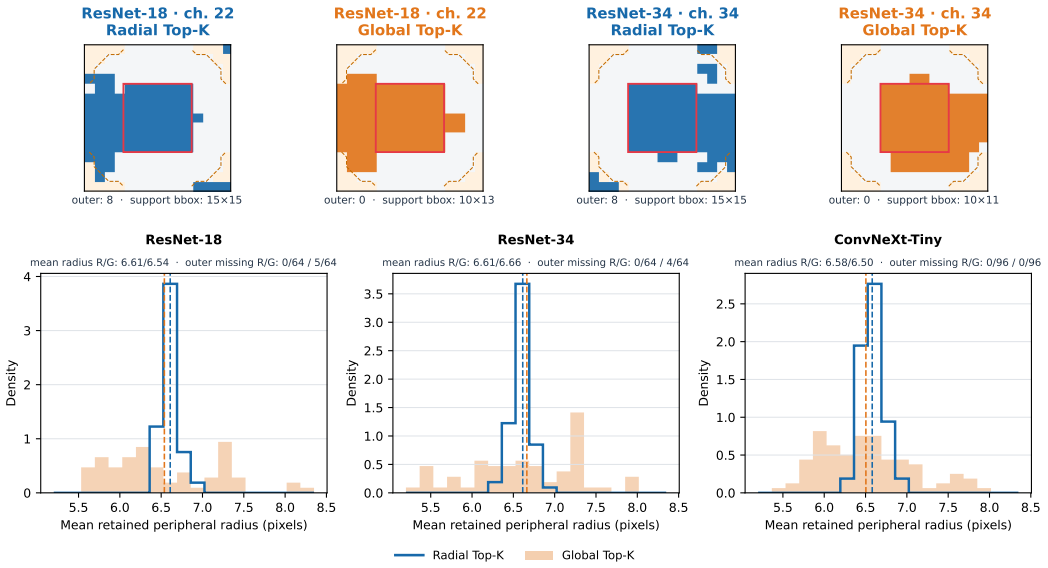


Figure 6: Radial Top-K and global Top-K selection with the same dense center and connection count. The top row shows representative ResNet masks and their support boxes. The bottom row shows mean retained peripheral-radius distributions across output channels for ResNet-18, ResNet-34, and ConvNeXt-Tiny; dashed vertical lines mark the corresponding means.

C.2 SPATIAL RESPONSE AND PERIPHERAL UTILIZATION

We examine peripheral utilization through input sensitivity and deployed weight energy. Figure 7 summarizes the normalized ERF maps of the 27 allocation models, each retaining 576 positions per output channel, by averaging responses within integer-radius bins. Unlike the two-dimensional differences in Figure 3, these profiles show the radial distribution itself, without subtracting a reference configuration. Responses peak near the center and extend with lower magnitude to larger radii. Figure 8 shows the deployed weight-energy composition across native RepLKNet stages. The protected center has the largest individual share in every displayed layer, while the combined peripheral share increases in later stages. Peripheral energy is therefore visible even where the native spatial kernel side becomes progressively smaller across stages.

C.3 INDIVIDUAL MASKS AND DIRECTIONAL ALLOCATION

We examine the spatial freedom that remains within the prescribed radial counts. Figure 9 shows output channel zero from the last target convolution of each stage. The dense center is preserved, and the selected peripheral positions vary across most displayed layers. The ConvNeXt-Tiny masks in stages 2–4, however, coincide with the deterministic flattened-index order used to break equal projection scores. In its deepest layers, the 15×15 kernel is comparable to or larger than the 14×14 and 7×7 feature maps, so many peripheral offsets sample predominantly or exclusively

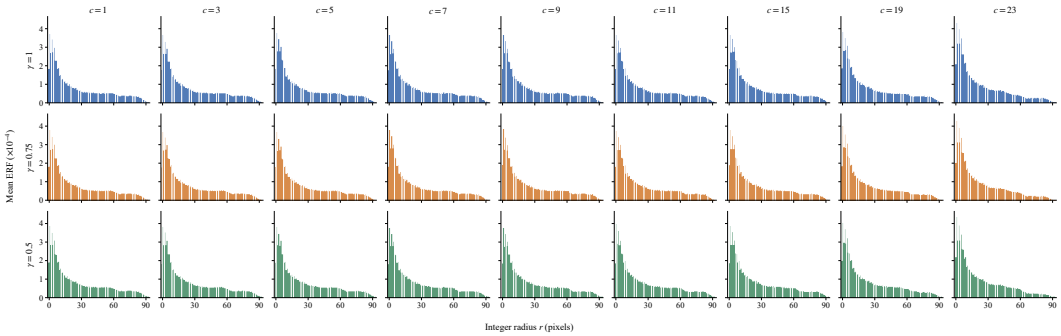


Figure 7: Mean normalized ERF within integer-radius bins for 27 fixed-count allocation models. Columns vary c and rows vary γ; values are radial responses rather than differences from a reference.

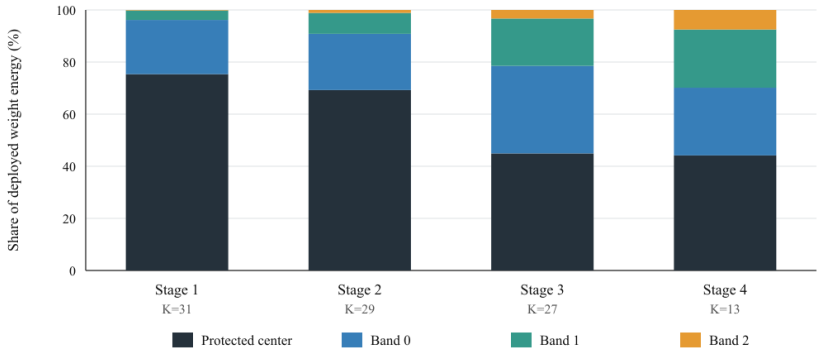


Figure 8: Deployed weight-energy composition across native RepLKNet stages, normalized independently within each layer.

padded zeros and receive little or no differentiated training signal. Their tied scores are consequently resolved by the fixed index rule; these repeated masks should be interpreted as a padding and tie-breaking effect rather than learned directionality. Figure 10 summarizes peripheral positions over all output channels in eight angular sectors. ResNet exhibits stage-dependent angular variation, while RepLKNet is closer to an even distribution. Radial counts permit different angular layouts when the learned scores distinguish positions, whereas the sector proportions also reflect feature-map support and discrete kernel geometry.

C.4 Variation in spatial utilization across channels

We inspect the variation hidden by averaging deployed weights over output channels. Figure 11 shows the distributions of energy-weighted normalized radius $\bar{r}_a$, defined in Appendix B.6, for three architecture checkpoints. ConvNeXt channels remain concentrated at relatively small radii across the displayed stages. RepLKNet develops a larger median radius in its final stage and a wider spread across channels. The ResNet distributions vary less and do not follow the same progression. Fixed band counts therefore permit channels to emphasize different portions of the available support. RepLKNet's broader final distribution associates greater peripheral weight utilization with increased channel variation.

C.5 Accuracy under changes in evaluation size

We evaluate the selected DTD checkpoints at input sizes from 160 to 320 to test the reuse of a fixed sparse topology. Figure 12 changes only deterministic resizing and center cropping. ResNet accuracy rises with input size before the curves flatten near the upper end of the range. Bio-RadDS stays above the displayed dense alternatives in both ResNet comparisons. ConvNeXt responds less after size 224 and shows some changes in method ordering. Increasing evaluation size therefore has a different effect across backbones. The learned sparse topology remains usable throughout the tested range, with the strongest accuracy response occurring in the ResNet models.

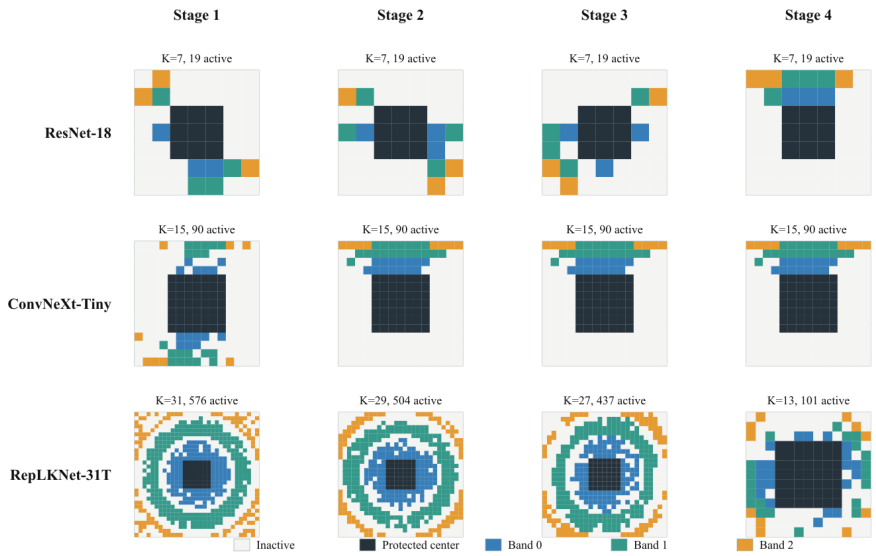


Figure 9: Representative spatial masks across architectures and stages. Each panel shows output channel zero of the last target convolution in the corresponding stage. Repeated deep ConvNeXt supports reflect padding-limited training signal and deterministic tie-breaking.

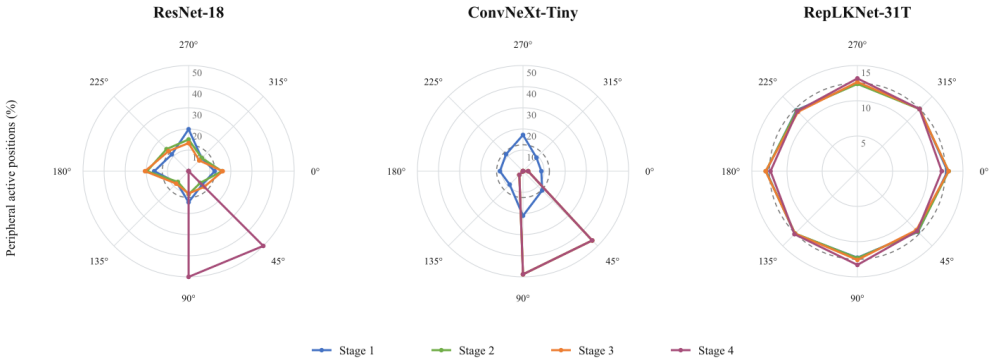


Figure 10: Angular allocation of retained peripheral positions, aggregated over all output channels into eight sectors. The circle indicates uniform allocation.

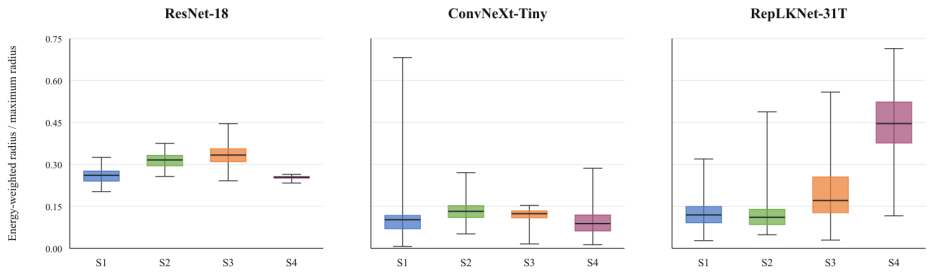


Figure 11: Energy-weighted normalized radius $\bar{r}_a$ across output channels, using squared deployed weights and center-to-corner radius normalization (Appendix B.6). Boxes show medians and interquartile ranges; whiskers span all channels in each stage.

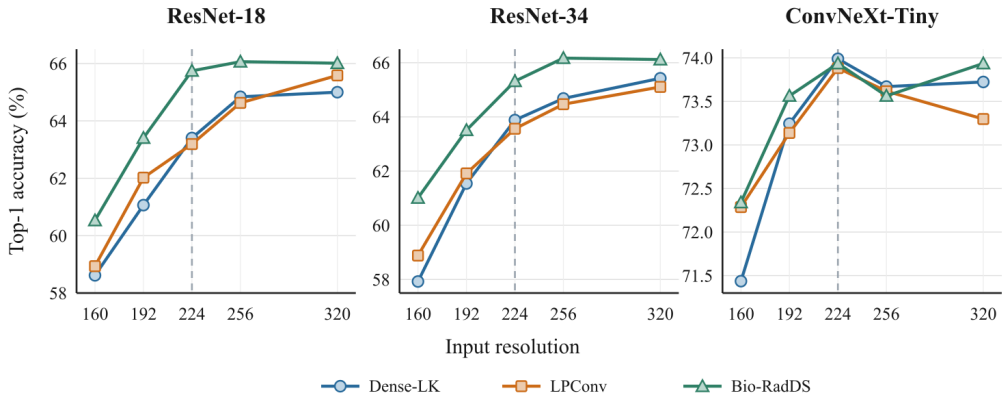


Figure 12: DTD accuracy at five evaluation sizes. Each curve uses a checkpoint selected at 224×224 with fixed weights and normalization statistics. Bio-RadDS also retains its selected topology throughout the sweep.

C.6 Accuracy and speed at the measured operating points

We compare the measured operating points by plotting the 15 accuracy and speedup pairs from Table 4 in Figure 13. The horizontal axis gives the target activity cap ρ, and each speedup uses the dense execution of that setting's own checkpoint. For ResNet-34, $\rho = 0.3$ achieves both the highest recorded accuracy and the largest speedup among the tested settings. For ConvNeXt-Tiny, the highest accuracy occurs at $\rho = 0.3$, whereas the largest speedup occurs at $\rho = 0.4$. ResNet-18 instead reaches its highest accuracy at $\rho = 0.4$, while $\rho = 0.3$ provides greater acceleration. Higher retention does not consistently improve accuracy and reduces ResNet speed. At $\rho = 0.7$, both ResNet variants become slower than their dense execution paths. ConvNeXt remains above $2.3\times$ throughout the sweep. These observations favor different retention settings according to the backbone and desired accuracy.

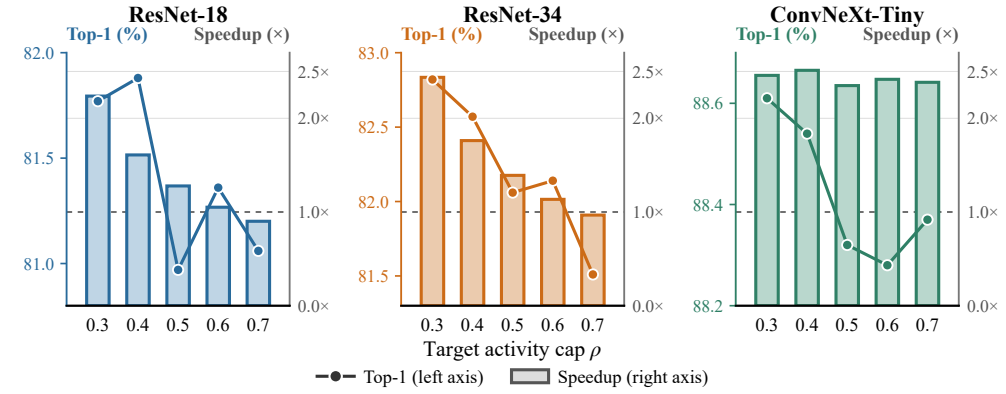


Figure 13: Accuracy (lines, left axes) and speedup (bars, right axes) from Table 4, plotted against the target activity cap ρ.

C.7 Numerical agreement after export

We test numerical agreement across the training representation, its folded dense form and compact sparse execution. Table 9 covers every target layer and the final logits of ResNet-18 and ConvNeXt-Tiny. The tests use random normal inputs supplemented by zero inputs and a central impulse. FP32 evaluation disables TF32 and uses absolute tolerance 10^{-5} with relative tolerance 10^{-4}. All 222 comparisons pass the combined tolerance test and produce finite values. Network predictions agree in every tested case. The largest absolute difference is 1.53×10^{-5}, and the largest relative L2 difference is 1.71×10^{-6}. For ConvNeXt, every tested output from the folded dense path is identical to its corresponding output from the compact sparse implementation.

Table 9: FP32 agreement between model representations. Counts include target layers and network outputs for three input cases. Errors are reported as maximum absolute difference and relative L2 difference.

Backbone	Comparison	Cases	Max abs.	Max rel. L2
ResNet-18	Training / folded	54	4.89×10^{-6}	1.20×10^{-6}
ResNet-18	Folded / sparse	54	1.53×10^{-5}	1.71×10^{-6}
ConvNeXt-Tiny	Training / folded	57	1.19×10^{-6}	5.96×10^{-7}
ConvNeXt-Tiny	Folded / sparse	57	0	0

C.8 Dense CUDA control for sparse acceleration

We compare four FP32 execution paths at 224×224 and batch size one. A is the independently trained Dense-LK checkpoint executed by cuDNN. B folds the Bio-RadDS mask into a zero-filled dense tensor and also uses cuDNN. C executes the same weights as B with dense Triton while visiting every spatial offset, and D uses compact sparse Triton to visit only retained offsets. A and B have nearly identical latency on both ResNet backbones. Under the matched Triton comparison, D is $2.394\times$ and $2.489\times$ faster than C on ResNet-18 and ResNet-34, respectively. This matched-backend comparison isolates the gain from skipping inactive offsets while holding the checkpoint and Triton implementation fixed.

Table 10: FP32 end-to-end latency (ms) at 224×224 on an RTX 3090 with batch size one (mean $\pm$ standard deviation over seven synchronized rounds). A uses Dense-LK; B–D use one Bio-RadDS checkpoint. A/B use cuDNN and C/D use dense/sparse Triton.

	A/B: cuDNN; C/D: Triton					
Backbone	A	B	C	D	A/D	C/D
ResNet-18	8.485 ± 0.007	8.474 ± 0.013	12.322 ± 0.045	5.148 ± 0.024	$1.648\times$	$2.394\times$
ResNet-34	18.336 ± 0.012	18.326 ± 0.013	25.898 ± 0.071	10.407 ± 0.035	$1.762\times$	$2.489\times$

C.9 FP16 inference across input sizes

We evaluate FP16 inference across input sizes to complement the FP32 study in the main paper. Table 11 measures ConvNeXt-Tiny on the RTX 3090 with batch size one. At 224×224, sparse execution takes 6.659 ms compared with 5.697 ms for Dense-LK and provides no acceleration. At 512×512, it reaches a speedup of $2.460\times$. The three larger inputs yield approximately fourfold acceleration, with the maximum of $4.304\times$ occurring at 768×768. Speedup therefore does not grow monotonically across the range. Peak memory stays close to the dense baseline and is sometimes slightly higher. The speed advantage is concentrated in the larger measured inputs.

Table 11: ConvNeXt-Tiny inference in FP16 with batch size one on RTX 3090. Latency reports the mean and standard deviation over five synchronized rounds. Speedup is measured relative to Dense-LK at each input size.

Size	Execution	MACs (G)	Latency (ms)	Memory (MiB)	Speedup
224	Dense-LK	4.832	5.697 ± 0.255	63.2	1.000
224	LPConv	4.832	5.703 ± 0.252	63.2	0.999
224	Ours	4.543	6.659 ± 0.273	63.8	0.855
512	Dense-LK	25.246	16.478 ± 0.093	88.0	1.000
512	LPConv	25.246	16.490 ± 0.065	88.0	0.999
512	Ours	23.733	6.697 ± 0.344	88.5	2.460
768	Dense-LK	56.804	38.014 ± 0.378	129.2	1.000
768	LPConv	56.804	37.934 ± 0.263	129.2	1.002
768	Ours	53.400	8.833 ± 0.207	129.8	4.304
1024	Dense-LK	100.985	63.384 ± 0.434	194.6	1.000
1024	LPConv	100.985	63.309 ± 0.335	194.6	1.001
1024	Ours	94.933	15.903 ± 0.239	182.5	3.986
2048	Dense-LK	403.940	258.260 ± 0.762	560.3	1.000
2048	LPConv	403.940	258.224 ± 0.694	560.3	1.000
2048	Ours	379.733	63.837 ± 0.242	560.9	4.046

C.10 VARIATION IN LATENCY ACROSS LAYERS

We replay the captured inputs of every target convolution through two deployment paths using the same folded Bio-RadDS checkpoint. Table 12 uses a conventional zero-filled PyTorch/cuDNN convolution as the dense reference and compact sparse Triton as the deployed operator. For ResNet-18, the sparse path is slower in the earliest layers, but becomes faster as channel depth increases and reaches about $4\times$ in the last stage. These are practical operator-level comparisons between cuDNN and Triton.

Table 12: Per-layer FP32 CUDA-event latency (ms) on RTX 3090 at 224^2 and batch size 1. The dense reference replays the folded Bio-RadDS weights as a zero-filled PyTorch/cuDNN convolution; the compact path uses sparse Triton. Speedup is cuDNN dense divided by sparse Triton. These isolated operator replays are not summed to estimate full-network latency.

Backbone	Stage and layer	Input shape	K	Groups	cuDNN dense	Sparse Triton	Speedup
ResNet-18	Stem: conv1	3×224^2	15	1	0.146	0.474	0.308
	S1: layer1.0.conv1	64×56^2	7	1	0.224	0.320	0.700
	S1: layer1.0.conv2	64×56^2	7	1	0.218	0.320	0.681
	S1: layer1.1.conv1	64×56^2	7	1	0.217	0.320	0.678
	S1: layer1.1.conv2	64×56^2	7	1	0.225	0.310	0.726
	S2: layer2.0.conv1	64×56^2	7	1	0.290	0.240	1.206
	S2: layer2.0.conv2	128×28^2	7	1	0.300	0.320	0.937
	S2: layer2.1.conv1	128×28^2	7	1	0.290	0.322	0.902
	S2: layer2.1.conv2	128×28^2	7	1	0.301	0.321	0.938
	S3: layer3.0.conv1	128×28^2	7	1	0.420	0.250	1.677
	S3: layer3.0.conv2	256×14^2	7	1	0.780	0.381	2.047
	S3: layer3.1.conv1	256×14^2	7	1	0.780	0.381	2.050
	S3: layer3.1.conv2	256×14^2	7	1	0.781	0.379	2.059
	S4: layer4.0.conv1	256×14^2	7	1	0.820	0.277	2.962
	S4: layer4.0.conv2	512×7^2	7	1	1.570	0.399	3.935
	S4: layer4.1.conv1	512×7^2	7	1	1.580	0.409	3.864
	S4: layer4.1.conv2	512×7^2	7	1	1.575	0.390	4.036