# KernelDNA: Cross-Layer Kernel Sharing via Decoupled Neural Adapters

**Yadong Zhang*** **Haiduo Huang*** **Yinghui Xu** **Tian Xia** **Wenzhe Zhao** **Pengju Ren†**
Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University

## Abstract

Dynamic convolution improves the representational flexibility of CNNs by generating input-adaptive kernels, yet existing methods either inflate parameters linearly with the number of base kernels or sacrifice inference throughput due to runtime kernel assembly. We revisit dynamic convolution from the angle of *cross-layer redundancy*: a Centered Kernel Alignment (CKA) analysis across modern CNN families shows that within-stage convolutional kernels are highly correlated, suggesting that an entire stage can be reparameterized by a single shared kernel together with lightweight, layer-specific transformations. Building on this observation, we propose **KernelDNA**, a parameter-efficient framework in which multiple convolutional layers in a stage share one "parent" kernel, while layer-specific "child" kernels are produced by a *decoupled* adapter that splits modulation into (i) an input-dependent dynamic channel gate and (ii) static spatial and filter modulations that can be pre-fused into the parent kernel before deployment. The factorization keeps every child weight identical across the batch, so each layer remains a standard convolution and avoids the per-sample kernel assembly that slows prior dynamic methods. We complement the design with a static approximation guarantee—when two kernels have squared-cosine alignment $\geq 1-\varepsilon$, a restricted static adapter recovers the target with relative Frobenius error at most $\sqrt{\varepsilon}$—and a conditional norm-budget comparison between shared and independent kernels, stating the scope of both explicitly. Across ImageNet-1K and MS-COCO, KernelDNA achieves strong accuracy–efficiency trade-offs over diverse backbones (ResNet18/50, MobileNetV2, ConvNeXt-Tiny), using fewer parameters than every baseline while retaining 90–99% of the standard-convolution throughput, and attaining the highest Top-1 on $4$ of $5$ backbones; the same factorization transfers to the attention and FFN projections of MobileViT-S and DeiT.

## 1 Introduction

Convolutional Neural Networks remain the workhorse of latency-critical visual computing, where their regular memory access pattern, linear complexity in the number of pixels, and built-in inductive biases continue to make them attractive on edge accelerators [He et al., 2016, Sandler et al., 2018, Liu et al., 2022]. A recurring tension in CNN design, however, is that a static kernel applies the *same* transformation to every input. **Dynamic convolution** [Yang et al., 2019, Chen et al., 2020] attacks this rigidity by aggregating $n$ static kernels with input-dependent attention weights $\pi_i(\mathbf{X})$, $\mathbf{W}(\mathbf{X}) = \sum_{i=1}^{n} \pi_i(\mathbf{X})\, \mathbf{W}_i$, trading additional parameters for representational adaptivity. This trade-off is steep: ODConv [Li et al., 2022] delivers a $+3.7\%$ Top-1 gain on ResNet18 at the cost of $\sim 4\times$ the parameters and $6.7\times$ lower GPU throughput; KernelWarehouse [Li and Yao, 2024] mitigates the parameter explosion by decomposing kernels into cells, but its runtime kernel assembly drags

*Equal contribution.
†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

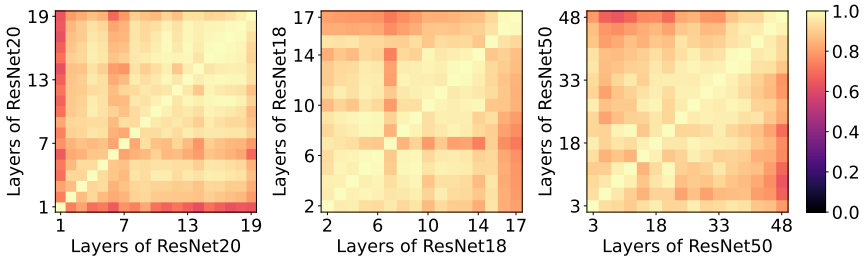


Figure 1: Linear CKA similarity among $3\times3$ convolutional kernels in pretrained ResNets at three depths (ResNet20, ResNet18, ResNet50; left to right). Only same-shape $3\times3$ kernels are compared, so stem, projection and downsampling layers are omitted. The strong block-diagonal structure persists across depths, indicating that stage-local kernels are highly redundant and motivating a shared-kernel reparameterization within each stage.

throughput down to $27\%$ of baseline on ResNet18; and FDConv [Chen et al., 2025] pushes accuracy via Fourier reparameterization but is $20\times$ slower at GPU inference on ResNet18.

These methods all treat each convolutional layer as an *independent* unit. We argue this is the wrong unit of design.

**A simple observation, often overlooked.** We compute the Linear Centered Kernel Alignment (CKA) [Kornblith et al., 2019] between every pair of same-shape $3\times3$ convolutional kernels within a stage of standard CNNs. As shown in Figure 1, kernels in the same stage exhibit pronounced block-diagonal similarity ($> 0.78$ on average) across ResNet variants at multiple depths. This is unsurprising in retrospect: within a stage, spatial resolution and channel widths are constant, the input distributions to consecutive blocks are statistically close due to residual connections, and SGD has no incentive to find substantially different solutions. What *is* surprising is that virtually no work in the dynamic convolution literature exploits this redundancy.

**Our proposal.** We introduce **KernelDNA**, a framework that turns this empirical regularity into an architecture. A single *parent* kernel is shared across multiple layers in a stage; each layer materializes its own *child* kernel through a lightweight adapter. The crux of the design is to **decouple** the adapter into two qualitatively different parts: **(i) static** per-layer spatial and filter modulators that are learned once and can be *absorbed* into the parent kernel before deployment, contributing zero runtime cost; and **(ii) dynamic** input-dependent channel gating that operates on the input feature map, leaving the convolution itself in standard format. The result, illustrated in Figure 2(d), is a *drop-in replacement* for standard convolutions that is more accurate than prior dynamic convolutions while retaining $90$–$99\%$ of the static baseline's throughput.

**What is new?** Cross-layer weight sharing has been studied for transformers [Lan et al., 2020, Bae et al., 2025, Zhang et al., 2022] and, statically, for CNNs [Boulch, 2017, Savarese and Maire, 2019]; input-dependent channel gates date back to SE-Net [Hu et al., 2018]; adapters appear in PEFT [Houlsby et al., 2019, Hu et al., 2022]. We do not claim any of these ingredients in isolation. Our contribution is a *constrained static–dynamic factorization* under which whole-kernel cross-layer sharing, layer-specific specialization, and input adaptivity coexist *without per-sample kernel assembly*: the static factors encode layer identity and are batch-independent, and the only dynamic factor is channel-wise, so by linearity it moves from the weight onto the input (Section 3). KernelWarehouse shares across layers and is input-adaptive but must materialize sample-specific weights; ShaResNet-style tying shares and keeps standard convolution but cannot adapt; SE gates inputs without any cross-layer kernel reparameterization. KernelDNA occupies the missing intersection, and same-depth controls show that neither sharing alone, gating alone, nor added depth alone reproduces its result (Section 5.4).

**Contributions.**

1. **A design principle for dynamic convolution.** We exploit empirically observed cross-layer redundancy to share one parent kernel across multiple layers in a stage, and show that a restricted static adapter recovers a target kernel with error controlled by its alignment to the parent (Theorem 1).

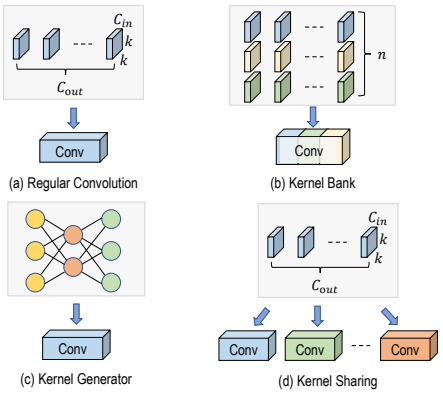


Figure 2: Paradigms of dynamic convolution. (a) Standard, fixed kernel. (b) *Kernel-pool* methods [Yang et al., 2019, Chen et al., 2020, Li et al., 2022, Chen et al., 2025, Li and Yao, 2024] maintain $n$ parallel kernels per layer, scaling parameters by $n\times$. (c) *Neural-generator* methods [Zhou et al., 2021, Li et al., 2021, He et al., 2023] synthesize kernels with auxiliary networks at inference. (d) *KernelDNA* (ours) shares one parent kernel across layers and materializes per-layer child kernels through a decoupled static–dynamic adapter, eliminating both parameter inflation and runtime assembly.

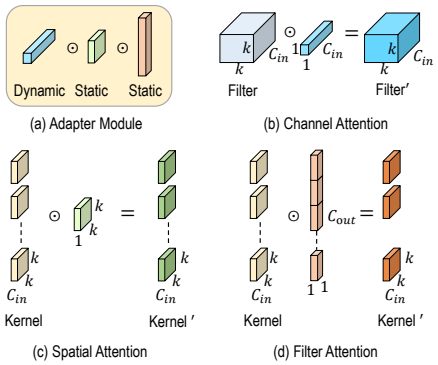


Figure 3: The KernelDNA decoupled adapter. A small SE block produces the dynamic channel gate $\beta_c$, applied to the input feature map; the static modulators $\alpha_s$ (spatial) and $\alpha_f$ (filter) are learned per child and combine multiplicatively with the shared parent kernel $\mathbf{W}$ to produce the child kernel $\widetilde{\mathbf{W}}^{(l)}$. The static modulation can be fused into the parent kernel at inference (latency-optimal) or kept separate (storage-optimal); in either case the convolution itself is a standard `F.conv2d`.

2. **Decoupled Neural Adapters (DNA).** A per-child adapter that splits modulation into (i) a dynamic SE-style *channel* gate—which by the linearity of convolution we apply equivalently to the input rather than to the kernel weight, and (ii) input-independent static *spatial* and *filter* modulators that produce the child kernel from the parent. The decoupling keeps the convolution weight identical across the batch, allowing every child layer to call a standard `F.conv2d` (with optional pre-fusion at deployment) and avoiding the batch-grouped convolutions that bottleneck prior dynamic methods (Section 3).
3. **Theoretical analysis with explicit scope.** (a) A static approximation guarantee under squared-cosine alignment; (b) a conditional comparison of worst-case norm budgets between shared and independent kernels (not a generalization bound); and (c) a gradient-SNR heuristic for the shared parent (Section 4).
4. **Strong accuracy–efficiency trade-offs.** On ResNet18, $74.23\%$ Top-1 with $9.24$ M params and $7{,}683$ fps—surpassing ODConv ($73.97\%$, $44.90$ M, $1{,}267$ fps), KernelWarehouse, and FDConv in both accuracy and speed; consistent gains hold across ResNet50, MobileNetV2, and ConvNeXt-Tiny on ImageNet-1K and MS-COCO, and the factorization transfers to the attention/FFN projections of MobileViT-S and DeiT (Section 5).

## 2 Related Work

**Dynamic convolution.** *Per-layer kernel-pool* methods—CondConv [Yang et al., 2019], DY-Conv [Chen et al., 2020], ODConv [Li et al., 2022], FDConv [Chen et al., 2025]—maintain $n$ static kernels at each layer and combine them through input-dependent attention, scaling parameters by $n\times$ *per layer*; a *neural-generator* family [Zhou et al., 2021, Li et al., 2021, He et al., 2023] instead synthesizes kernels with auxiliary networks at inference time. KernelWarehouse [Li and Yao, 2024] is the closest precedent in spirit: it shares a pool of fine-grained kernel *cells* across multiple layers and assembles each layer's kernel from these cells via *input-dependent* attention, achieving genuine parameter compression—but the input-dependent assembly produces a different kernel for every batch element, forcing a batch-grouped or accumulated convolution that drags throughput to 26–$41\%$ of baseline depending on architecture (Table 1). KernelDNA differs along two axes: it shares *entire parent kernels* (a much coarser granularity than sub-kernel cells), and—critically—its static modulators $(\alpha_s, \alpha_f)$

are *input-independent*, so each child kernel is identical across the batch and the layer reduces to a standard `F.conv2d`; the optional pre-fusion at deployment merely eliminates a small (kernel-sized, batch-independent) element-wise multiplication.

**Cross-layer weight sharing in CNNs.** ShaResNet [Boulch, 2017] ties identical weights across residual blocks but applies no adaptation, sacrificing capacity. Soft sharing through implicit-recurrence templates [Savarese and Maire, 2019] introduces scalar-coefficient mixtures of a small basis, and Neural Parameter Allocation Search [Plummer et al., 2022] automates parameter assignment across layers. None of these methods provides input-adaptive modulation; KernelDNA augments structured sharing with a dynamic channel gate, achieving both parameter compression and the adaptivity of dynamic convolution. The transformer counterparts ALBERT [Lan et al., 2020], MiniViT [Zhang et al., 2022], and Relaxed Recursive Transformers [Bae et al., 2025] share weights across layers and report regularization benefits; our work adapts this principle to convolutional architectures with attention-modulated kernels, and we show in Section 5.3 that the same factorization applies to transformer projections.

**Visual attention, adapters, and orthogonal compression.** SE-Net [Hu et al., 2018], CBAM [Woo et al., 2018], ECA [Wang et al., 2020], and CGC [Lin et al., 2020] add attention *alongside* convolutions; KernelDNA instead injects attention *into* the kernel-generation pathway, an idea reminiscent of LoRA [Hu et al., 2022] but applied to convolutional reparameterization rather than fine-tuning. Pruning [Li et al., 2017], low-rank factorization [Denton et al., 2014], and quantization [Esser et al., 2020] are orthogonal to KernelDNA and can in principle be combined with it (Section 6); deformable convolutions [Dai et al., 2017, Zhu et al., 2019, Xiong et al., 2024] adapt *where* to sample whereas dynamic convolution adapts *how* to transform—the two are complementary, and we focus on the latter with a sharing-centric solution.

# 3 Method: KernelDNA

## 3.1 Preliminaries

A standard convolution applies a filter $\mathbf{W} \in \mathbb{R}^{c' \times c \times k \times k}$ to an input feature map $\mathbf{X} \in \mathbb{R}^{c \times h \times w}$, producing $\mathbf{Y}_{m,i,j} = \sum_{n,p,q} \mathbf{W}_{m,n,p,q} \mathbf{X}_{n,i+p,j+q} + \mathbf{b}_m$. Dynamic convolution [Yang et al., 2019] replaces $\mathbf{W}$ with $\mathbf{W}(\mathbf{X}) = \sum_{i=1}^{n} \pi_i(\mathbf{X}) \mathbf{W}_i$, where $\sum_i \pi_i = 1$. Each per-layer kernel pool has $nc'ck^2$ parameters and requires either a memory-intensive batch-grouped convolution (ODConv) or runtime kernel assembly (KernelWarehouse, FDConv) at inference.

## 3.2 Cross-Layer Kernel Sharing

Let $\{\widetilde{\mathbf{W}}^{(l)}\}_{l=1}^{L}$ denote the kernels of $L$ convolutional layers within a stage. Motivated by the stage-local redundancy in Figure 1, we parameterize each layer-specific child kernel by a shared parent and a structured multiplicative adapter,

$$\widetilde{\mathbf{W}}^{(l)} = \mathbf{W} \odot \Gamma^{(l)}, \qquad \Gamma^{(l)} = \beta_c^{(l)}(\mathbf{X}) \odot \big(1+\alpha_f^{(l)}\big) \odot \big(1+\alpha_s^{(l)}\big), \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{c' \times c \times k \times k}$ is a single shared *parent* kernel and $\Gamma^{(l)}$ is the layer-specific multiplicative *modulation* produced by the adapter (illustrated in Figure 3). Here $\odot$ denotes a *broadcast* (not strictly element-wise) product: the three factors have shapes $\beta_c^{(l)}(\mathbf{X}) \in (0,1)^{1 \times c \times 1 \times 1}$ (input channels), $\alpha_f^{(l)} \in \mathbb{R}^{c' \times 1 \times 1 \times 1}$ (output filters), and $\alpha_s^{(l)} \in \mathbb{R}^{1 \times 1 \times k \times k}$ (spatial offsets), each expanded along its singleton axes to the kernel shape. The modulation axes are inspired by multi-dimensional dynamic convolution such as ODConv, but Equation (1) is a deliberately *constrained* form: only the channel factor depends on the input, and it is the one axis that can be moved out of the kernel by linearity.

**Design principle (why decouple?).** By the linearity of convolution, $\mathbf{W} * (\beta_c \odot \mathbf{X}) = (\mathbf{W} \odot \beta_c) * \mathbf{X}$ when $\beta_c$ varies only along the contracted input-channel axis, so the dynamic factor can be moved *out of the kernel weight onto the input feature map*, leaving only the input-independent static modulation in the kernel. Hence the convolution weight is identical across the batch, every child layer reduces to a standard `F.conv2d`, and we sidestep the batch-grouped or per-sample-assembled convolutions that dominate the latency of ODConv and KernelWarehouse (Table 1). An input-dependent *mixture* of kernels (as in kernel pools or KernelWarehouse cells) cannot be moved to the input this way, which

is why those methods must materialize sample-specific weights. KernelDNA is therefore *dynamic* in the sense of dynamic convolution—the effective transformation depends on the input through $\beta_c(\mathbf{X})$—while the factors that encode *layer identity* are static. We now detail the three components.

**Dynamic channel gate $\beta_c^{(l)} \in (0,1)^c$.** A modified Squeeze-and-Excitation block [Hu et al., 2018]—global average pool followed by two $1\times1$ convolutions with an intervening BatchNorm [Ioffe and Szegedy, 2015] and ReLU, and a sigmoid output—produces an input-dependent gate vector that is applied directly to the input feature map (per the identity above), so the dynamic component never enters the kernel weight. In trained models the gate is genuinely input-dependent, not a nearly static vector (Appendix D.6).

**Static spatial modulator $\alpha_s^{(l)} \in \mathbb{R}^{1\times1\times k\times k}$ and filter modulator $\alpha_f^{(l)} \in \mathbb{R}^{c'\times1\times1\times1}$.** These are learnable parameters initialized to zero, so that at the start of training the child kernel equals the parent kernel. Crucially, both modulators are *input-independent*: they encode *layer identity*, not input content, with two consequences. First, at deployment they *can be* fused into the parent kernel, $\mathbf{W}_{\text{fused}}^{(l)} = \mathbf{W} \odot (1+\alpha_f^{(l)}) \odot (1+\alpha_s^{(l)})$, yielding a standard convolution kernel with no runtime modulation cost—the *latency-optimal* mode, in which each child stores its own fused kernel and parent–child storage sharing is given up for the fused layers. Alternatively, the parent and the small modulators can be kept and the fusion performed at runtime at the cost of one $O(c'ck^2)$ element-wise product *per layer per forward pass*, independent of batch size and resolution—the *storage-optimal* mode, which preserves the parameter counts we report. The two modes are endpoints of a storage–latency trade-off for one trained model; throughout, parameter counts refer to the storage-optimal representation and throughput/latency are measured in the fused mode (Appendix B). Second, only the channel gate remains input-adaptive at inference, focusing the dynamic budget on the dimension where it provides the most empirical benefit (Section 5.4).

### 3.3 Placement and Same-Depth Controls

Within each ResNet stage, two consecutive BasicBlocks contain four $3\times3$ convolutions; letting `F` denote a full (parent) kernel and `S` a shared (child) layer, we examined every reasonable arrangement (Table 7). The pattern `FS-SF-SF`, with two parents and three children per stage, achieves the best accuracy/parameter trade-off and is adopted in stages 3 and 4 of the ResNets. For ResNet50 we share only the bottleneck $3\times3$ kernel; for MobileNetV2 we share only the pointwise $1\times1$ kernels of inverted bottlenecks (depthwise $3\times3$ kernels are too small to compress meaningfully); for ConvNeXt-Tiny we share the depthwise $7\times7$ kernels. Placement is hand-designed; the rule we followed (share only repeated, same-role, same-shape operators within a stage; keep transition/downsampling operators independent; at most three children per parent) and the per-backbone layouts are given in Appendix B. Note that `FS-SF-SF` adds physical conv layers vs. the original BasicBlock (*e.g.*, ResNet18+KernelDNA has 22 conv layers vs. 18); *same-depth* controls with independent (`Expand`) and unadapted shared (`Copy`) kernels at the same positions are reported in Section 5.4.

**Algorithmic comparison.** The decoupled design avoids two bottlenecks of dynamic convolution: *(i)* batch-expanded weight tensors of shape $[B, N, c', c, k, k]$ and *(ii)* batch-grouped convolutions. Child kernels are deterministic functions of $(\mathbf{W}, \alpha_s, \alpha_f)$ and the convolution is standard `F.conv2d`, which is why KernelDNA retains 90–99% of static-convolution throughput while every compared dynamic method retains under 60% (CondConv, ODConv, FDConv below 30% on ResNet18; Table 1); a side-by-side PyTorch listing is in Appendix A.

## 4 Theoretical Analysis

We provide three results of deliberately different status: (i) a *static* approximation guarantee showing that a restricted member of the adapter family recovers a target kernel with error controlled by its alignment to the parent (Theorem 1); (ii) a *conditional* comparison of worst-case norm budgets between a shared parameterization and independent kernels (Theorem 2), a proxy motivated by norm-based capacity bounds and *not* itself a generalization bound; and (iii) a gradient-SNR *heuristic* for the shared parent. Full proofs are in Appendix C; none of the results concerns the trained input-dependent gate, whose behavior is studied empirically in Appendix D.6.

**Notation.** For kernels $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{c' \times c \times k \times k}$ viewed as vectors in $\mathbb{R}^{c'ck^2}$, define the squared-cosine alignment

$$\mathrm{ALIGN}(\mathbf{W}_1, \mathbf{W}_2) = \frac{\langle \mathbf{W}_1, \mathbf{W}_2 \rangle^2}{\|\mathbf{W}_1\|_F^2 \, \|\mathbf{W}_2\|_F^2} \in [0, 1]. \tag{2}$$

ALIGN coincides with the *uncentered* linear CKA of Kornblith et al. [2019] when each kernel is treated as a single vector; it is an entrywise statistic and differs from the centered, Gram-matrix CKA plotted in Figure 1, which we use only as motivation; Theorem 1 is conditional on ALIGN. The *adapter family* $\mathcal{A}$ consists of triples $(\beta_c, \alpha_f, \alpha_s)$ with $\beta_c$ produced by the sigmoid-output SE gate and $\alpha_f \in \mathbb{R}^{c'}$, $\alpha_s \in \mathbb{R}^{k \times k}$ unconstrained; constant gates $\beta_c \equiv b\mathbf{1}_c$, $b \in (0,1)$, are realizable (zero the last SE layer's weights and set its bias to $\log \frac{b}{1-b}$). $\|\cdot\|_\infty$ is the largest absolute entry of a tensor after broadcasting to the kernel shape, and $\mathbf{1}$ the all-ones tensor.

### 4.1 Static Approximation under Squared-Cosine Alignment

**Theorem 1** (Static approximation under alignment)**.** *Let $\mathbf{W}, \mathbf{W}^* \in \mathbb{R}^{c' \times c \times k \times k}$ be nonzero with $\mathrm{ALIGN}(\mathbf{W}, \mathbf{W}^*) \geq 1 - \varepsilon$ for some $\varepsilon \in [0,1]$, and let $c = \langle \mathbf{W}, \mathbf{W}^* \rangle / \|\mathbf{W}\|_F^2$. For every $b \in (0,1)$ the static member $\beta_c \equiv b\mathbf{1}_c$, $\alpha_f = (c/b - 1)\mathbf{1}_{c'}$, $\alpha_s = \mathbf{0}$ of $\mathcal{A}$ yields $\widetilde{\mathbf{W}} = \mathbf{W} \odot \beta_c \odot (1+\alpha_f) \odot (1+\alpha_s) = c\,\mathbf{W}$ and*

$$\big\|\widetilde{\mathbf{W}} - \mathbf{W}^*\big\|_F \;=\; \sqrt{1 - \mathrm{ALIGN}(\mathbf{W}, \mathbf{W}^*)}\, \|\mathbf{W}^*\|_F \;\leq\; \sqrt{\varepsilon}\, \|\mathbf{W}^*\|_F. \tag{3}$$

*Sketch.* Decompose $\mathbf{W}^* = c\mathbf{W} + \mathbf{W}_\perp$ with $\langle \mathbf{W}, \mathbf{W}_\perp \rangle = 0$. By Pythagoras, $\|\mathbf{W}_\perp\|_F^2 = \|\mathbf{W}^*\|_F^2 - c^2\|\mathbf{W}\|_F^2 = (1 - \mathrm{ALIGN})\|\mathbf{W}^*\|_F^2$. The chosen factors multiply to the constant $b \cdot (c/b) = c$, so $\widetilde{\mathbf{W}} = c\mathbf{W}$ and the residual is exactly $\mathbf{W}_\perp$. □

*Remark* 1 (Scope of Theorem 1). (i) The theorem is an *existence* statement about a restricted static subclass of $\mathcal{A}$; the full family (for any fixed input, hence any fixed gate value) approximates at least as well, but nothing is said about which member training selects. (ii) It is a *static* result: when the optimal kernel depends on the input ($\mathbf{W}^* = \mathbf{W}^*(\mathbf{X})$) it provides no uniform input-conditional bound; a distribution-dependent analysis of the trained gate is left to future work. Empirically, the channel gate is the most valuable adapter component (Table 8, $+1.97$ Top-1 over sharing alone) and is substantially input-dependent after training (Appendix D.6); we keep this evidence separate from the theorem. (iii) Nothing is assumed about the entries of $\mathbf{W}$ (zeros are allowed); the bound is informative when $\varepsilon$ is small.

### 4.2 Worst-Case Norm Budget of Shared vs. Independent Kernels

We compare two parameterizations of $N \geq 2$ same-shape kernels $\mathbf{K}_1, \ldots, \mathbf{K}_N$ in one stage. *Independent:* $\mathbf{K}_l = \mathbf{W}_l$ with $\|\mathbf{W}_l\|_F \leq B_W$. *Shared (KernelDNA):* $\mathbf{K}_1 = \mathbf{W}$ is the parent layer and, for $2 \leq l \leq N$, $\mathbf{K}_l = \mathbf{W} \odot \Gamma^{(l)} = \mathbf{W} + \mathbf{W} \odot \delta^{(l)}$ with $\delta^{(l)} := \Gamma^{(l)} - \mathbf{1}$, under $\|\mathbf{W}\|_F \leq B_W$ and $\sup_{\mathbf{X}} \|\delta^{(l)}(\mathbf{X})\|_\infty \leq B_\theta$. The identity $\mathbf{W} \odot \Gamma^{(l)} = \mathbf{W} + \mathbf{W} \odot \delta^{(l)}$ is purely algebraic (child = parent + effective perturbation); the perturbation is *not* an independently trainable tensor. Define the *decomposition budget* as the sum of Frobenius norms of the tensors in each decomposition, $B_{\text{indep}} = \sum_{l=1}^{N} \|\mathbf{W}_l\|_F$ and $B_{\text{KDNA}} = \|\mathbf{W}\|_F + \sum_{l=2}^{N} \sup_{\mathbf{X}} \|\mathbf{W} \odot \delta^{(l)}(\mathbf{X})\|_F$, and the *worst-case budgets* $\bar{B}_{\text{indep}}, \bar{B}_{\text{KDNA}}$ as their suprema over the respective constraint sets. Norm-based capacity bounds [Bartlett and Mendelson, 2002, Golowich et al., 2018] grow with such weight norms, which is our motivation for the proxy; we do not derive a Rademacher-complexity or generalization bound.

**Theorem 2** (Worst-case decomposition budget)**.** *$\bar{B}_{indep} = NB_W$ and $\bar{B}_{KDNA} = \big(1 + (N-1)B_\theta\big)B_W$. Consequently $\bar{B}_{KDNA}/\bar{B}_{indep} = \big(1 + (N-1)B_\theta\big)/N$, which is $< 1$ if and only if $B_\theta < 1$.*

*Sketch.* $\bar{B}_{\text{indep}} = NB_W$ is attained at $\|\mathbf{W}_l\|_F = B_W$. For the shared class, $\|\mathbf{W} \odot \delta\|_F \leq \|\mathbf{W}\|_F \|\delta\|_\infty \leq B_W B_\theta$, so $\bar{B}_{\text{KDNA}} \leq (1 + (N-1)B_\theta)B_W$, with equality at $\delta^{(l)} \equiv B_\theta \mathbf{1}$ (realizable in $\mathcal{A}$). Dividing exact suprema gives the ratio. □

*Remark* 2. For a parent with two children ($N = 3$) and $B_\theta = 0.1$, the worst-case budget of the shared parameterization is $0.4\times$ that of three independent kernels. This quantifies how much "effective weight" sharing removes within the norm-constrained subclass; it is not a statement about test error and does not count the adapter parameters themselves.

### 4.3 Gradient Flow: A Signal-to-Noise Heuristic

The shared parent $\mathbf{W}$ receives a gradient that aggregates contributions from the $L$ layers that use it (the parent layer itself and its children):

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{l=1}^{L} \left(1+\alpha_f^{(l)}\right) \odot \left(1+\alpha_s^{(l)}\right) \odot \mathbf{g}^{(l)}, \qquad \mathbf{g}^{(l)} = \partial \mathcal{L}/\partial \mathbf{W}_{\text{fused}}^{(l)}, \tag{4}$$

where $\mathbf{W}_{\text{fused}}^{(l)}$ is the static child kernel of Section 3.2 (the gate acts on the input, so its effect is contained in $\mathbf{g}^{(l)}$). Model $\{\mathbf{g}^{(l)}\}$ as random vectors over mini-batch sampling with a *common* mean $\mu$, common variance $\sigma^2$ (coordinate-wise), and pairwise correlation $\rho \in [0, 1]$, and take the static modulation to be $\mathbf{1}$ (exact at initialization, where $\alpha_s, \alpha_f = 0$). Then the parent-gradient signal-to-noise ratio is

$$\text{SNR}_{\mathbf{W}} = \frac{(L\mu)^2}{L\sigma^2(1+(L-1)\rho)} = \frac{L\,\mu^2/\sigma^2}{1+(L-1)\rho} \;\geq\; \mu^2/\sigma^2, \tag{5}$$

with equality only at $\rho = 1$: independent child gradients ($\rho \to 0$) average out noise and amplify SNR by $L$, perfectly correlated ones leave SNR unchanged while reinforcing the shared direction. We measured $\rho > 0.6$ between the three gradient contributions to one ResNet18 stage-3 parent kernel ($L = 3$: its own layer and its two children) throughout training (Appendix C.3). Two caveats: the common-mean assumption fails once children *specialize*—child-specific disagreement then partially cancels in the parent's gradient, and the per-child adapter parameters $\theta^{(l)} = (\alpha_s^{(l)}, \alpha_f^{(l)}, \text{SE}^{(l)})$, whose gradients depend only on their own child's loss path, are intended to absorb it; and our ablations establish *that* adapter-mediated sharing helps, not *which* of regularization, conditioning, or gradient averaging is responsible. We present Equation (5) as intuition, not as an established explanation.

## 5 Experiments

We evaluate KernelDNA on (i) image classification on ImageNet-1K [Russakovsky et al., 2015], (ii) object detection and instance segmentation on MS-COCO [Lin et al., 2014] with Mask R-CNN [He et al., 2017], (iii) an extension to the attention/FFN projections of MobileViT-S [Mehta and Rastegari, 2022] and DeiT [Touvron et al., 2021], and (iv) ablations of placement, adapter components, channel ratios, parameter scaling, and training cost. Throughput and latency are measured on a single Nvidia RTX 4090 GPU (batch 128) and an AMD EPYC 7H12 CPU (batch 1, single thread), averaged over 1,000 iterations, with KernelDNA in the fused (latency-optimal) mode. **Every CNN method (baselines included) is trained from scratch under the identical training protocol of KernelWarehouse** [Li and Yao, 2024]: SGD (100 epochs, batch 256) for ResNet/MobileNetV2; AdamW (300 epochs, batch 1024) for ConvNeXt-Tiny. **Seeds.** Accuracies in Tables 1 and 2 are single runs (seed 42), following the compared methods. Two additional KernelDNA seeds give $74.23/74.20/74.24$ ($74.22 \pm 0.02$) on ResNet18 and $75.51/75.47/75.50$ ($75.49 \pm 0.02$) on MobileNetV2-1.0× (mean±sample std, Table 5). Because the baselines are single runs, we make no significance claim for the smallest gaps (*e.g.*, $+0.09$ to $+0.26$); parameter counts and operator-level throughput are seed-independent.

### 5.1 Image Classification on ImageNet-1K

Table 1 presents our main results. KernelDNA delivers the highest Top-1 on 4 of 5 backbones while using *fewer parameters than the static baseline and every dynamic-convolution method* and retaining 90–99% of the baseline GPU throughput. (i) **vs. ODConv on ResNet18**: 74.23% vs. 73.97% Top-1 with 4.9× fewer parameters and 6× higher throughput. (ii) **vs. KernelWarehouse on ResNet50**: 80.95% vs. 80.38%, 1.2× fewer parameters, 2.5× higher throughput, lower FLOPs. (iii) **vs. FDConv on ResNet18/50**: +2.2 to +3.0% Top-1 at 10–18× higher GPU throughput; the Fourier-domain reparameterization does not pay off on standard hardware. (iv) **MobileNetV2 (1.0×)**: 75.51% Top-1 with *fewer parameters than the baseline* (3.14 vs. 3.50 M), so attention-modulated sharing can compress and improve a model simultaneously. The single Top-1 loss is on MobileNetV2-0.5×, where ODConv reaches 70.01% vs. our 69.32% at 2.4× our parameters and 6.0× our CPU latency (KernelDNA still leads on Top-5). Relative to the static baselines, KernelDNA costs 1–9% GPU throughput and 2–28% CPU latency; it is *not* faster than the original CNNs, and its speed advantage is specifically over dynamic-convolution methods.

Table 1: ImageNet-1K classification results. KDNA = KernelDNA (ours). Pm: parameters (M); FP: FLOPs (G); TP-GPU: throughput (fps, batch 128); Lt-CPU: single-thread latency (ms, batch 1); Top-1/Top-5 accuracy (%). **Bold**: best value per column within a backbone group, static baseline included (ties all bold). KernelDNA has the fewest parameters and the highest Top-1 on $4$ of $5$ backbones, and retains throughput closest to the static baseline among all dynamic-convolution variants.

| Model | Pm (M) | FP (G) | TP-GPU (fps) | Lt-CPU (ms) | Top-1 (%) | Top-5 (%) |
|---|---|---|---|---|---|---|
| ResNet18 | 11.69 | **1.82** | **8463** | **22.16** | 70.25 | 89.38 |
| + SE [Hu et al., 2018] | 11.78 | **1.82** | 8319 | 23.68 | 70.98 | 90.03 |
| + CBAM [Woo et al., 2018] | 11.78 | **1.82** | 6023 | 25.25 | 71.01 | 89.85 |
| + ECA [Wang et al., 2020] | 11.69 | **1.82** | 8372 | 23.54 | 70.60 | 89.68 |
| + CGC [Lin et al., 2020] | 11.69 | 2.19 | 1537 | 78.92 | 71.60 | 90.35 |
| + CondConv [Yang et al., 2019] | 81.35 | **1.82** | 280 | 89.90 | 71.99 | 90.27 |
| + DY-Conv [Chen et al., 2020] | 45.47 | 1.87 | 3308 | 46.15 | 72.76 | 90.79 |
| + ODConv [Li et al., 2022] | 44.90 | **1.82** | 1267 | 54.34 | 73.97 | 91.35 |
| + KW [Li and Yao, 2024] | 11.93 | 2.23 | 2272 | 69.62 | 73.67 | 91.17 |
| + FDConv [Chen et al., 2025] | 11.75 | 1.92 | 422 | 1013 | 71.25 | 90.04 |
| + **KDNA (ours)** | **9.24** | 2.28 | 7683 | 28.43 | **74.23** | **92.38** |
| ResNet50 | 25.56 | **4.11** | **2432** | **90.77** | 78.44 | 94.24 |
| + DY-Conv [Chen et al., 2020] | 100.88 | 4.22 | 1161 | 155.63 | 79.00 | 94.27 |
| + ODConv [Li et al., 2022] | 90.67 | 4.12 | 560 | 132.39 | 80.62 | 95.16 |
| + KW [Li and Yao, 2024] | 28.05 | 6.15 | 922 | 138.78 | 80.38 | 95.19 |
| + FDConv [Chen et al., 2025] | 27.16 | 4.29 | 230 | 1380.65 | 78.76 | 94.35 |
| + **KDNA** | **22.78** | 4.33 | 2346 | 93.12 | **80.95** | **95.54** |
| MobileNetV2 (0.5×) | 1.97 | **0.10** | **12787** | **10.04** | 64.30 | 85.21 |
| + DY-Conv [Chen et al., 2020] | 4.57 | 0.11 | 5818 | 45.59 | 69.05 | 88.37 |
| + ODConv [Li et al., 2022] | 4.44 | 0.11 | 3761 | 62.07 | **70.01** | 89.01 |
| + KW [Li and Yao, 2024] | 2.85 | 0.16 | 3284 | 26.43 | 68.29 | 87.93 |
| + **KDNA** | **1.88** | 0.11 | 12461 | 10.27 | 69.32 | **89.04** |
| MobileNetV2 (1.0×) | 3.50 | **0.31** | **6323** | **30.21** | 72.02 | 90.43 |
| + DY-Conv [Chen et al., 2020] | 12.40 | 0.33 | 3667 | 46.93 | 74.94 | 91.83 |
| + ODConv [Li et al., 2022] | 11.52 | 0.32 | 2226 | 59.51 | 75.42 | 92.18 |
| + KW [Li and Yao, 2024] | 5.17 | 0.47 | 2288 | 44.58 | 74.68 | 91.90 |
| + **KDNA** | **3.14** | **0.31** | 6260 | 33.46 | **75.51** | **92.27** |
| ConvNeXt-Tiny | 28.59 | **4.47** | **1847** | **56.20** | 82.07 | 95.86 |
| + KW [Li and Yao, 2024] | 32.99 | 5.61 | 752 | 224.98 | 82.55 | 96.08 |
| + **KDNA** | **24.32** | 4.51 | 1739 | 60.12 | **82.76** | **96.34** |

### 5.2 Object Detection and Instance Segmentation on MS-COCO

We pretrain backbones on ImageNet-1K and fine-tune Mask R-CNN with the standard $1\times$ schedule via MMDetection [Chen et al., 2019]. Table 2 reports detection and segmentation AP across three backbones. KernelDNA achieves the best AP under all settings, including where ConvNeXt-Tiny+FDConv is competitive, and the gain is uniform across object sizes, suggesting that the multiplicative adapter does not bias the model toward any spatial scale.

### 5.3 Extension to Transformer Projections

A linear projection is a $1\times1$ convolution over tokens, so the same parent–child factorization applies to attention and FFN weight matrices with the spatial modulator omitted and the channel gate pooled over tokens. We share the QKV, output, FFN-in and FFN-out projections within each transformer stage of MobileViT-S (a convolution–attention hybrid) and DeiT-Tiny/Small; each child keeps its own normalization, biases, $\alpha_f$, and gate (Appendix D.5). Table 4 compares against KernelWarehouse, extended to these backbones by importing its warehouse mechanism unmodified into its published DeiT recipe; MobileViT-S rows use $256^2$ inputs and a $150$-epoch recipe for all methods, DeiT rows use $224^2$ (published DeiT-Tiny/Small baselines: $72.2/79.8$ [Touvron et al., 2021]). On MobileViT-S, KernelDNA offers an *iso-accuracy* point (4P/5C: $77.97\%$ vs. KW's $77.94\%$ with $30\%$ fewer parameters and $2.2\times$ the throughput), an *iso-parameter* point (18 blocks: $+0.87$ Top-1 and $1.7\times$ the throughput at $0.9\%$ more parameters), and a compression point (3P/6C: $-33\%$ parameters,

Table 2: MS-COCO detection and instance segmentation with Mask R-CNN (1× schedule). Best value per column within each backbone group in bold. KernelDNA outperforms all dynamic convolution baselines on all backbones.

| | Object Detection | | | | | | Instance Segmentation | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Backbone | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
| ResNet50 | 39.6 | 61.6 | 43.3 | 24.4 | 43.7 | 50.0 | 36.4 | 58.7 | 38.6 | 20.4 | 40.4 | 48.4 |
| + DY-Conv [Chen et al., 2020] | 39.6 | 62.1 | 43.1 | 24.7 | 43.3 | 50.5 | 36.6 | 59.1 | 38.6 | 20.9 | 40.2 | 49.1 |
| + ODConv [Li et al., 2022] | 42.1 | 65.1 | 46.1 | 27.2 | 46.1 | 53.9 | 38.6 | 61.6 | 41.4 | 23.1 | 42.3 | 52.0 |
| + KW [Li and Yao, 2024] | 41.8 | 64.5 | 45.9 | 26.6 | 45.5 | 53.0 | 38.4 | 61.4 | 41.2 | 22.2 | 42.0 | 51.6 |
| + **KDNA** | **42.5** | **66.2** | **46.5** | **28.0** | **46.5** | **54.5** | **39.1** | **61.9** | **41.5** | **23.5** | **42.8** | **52.6** |
| MobileNetV2 (1.0×) | 33.8 | 55.2 | 35.8 | 19.7 | 36.5 | 44.4 | 31.7 | 52.4 | 33.3 | 16.4 | 34.4 | 43.7 |
| + DY-Conv [Chen et al., 2020] | 37.0 | 58.6 | 40.3 | 21.9 | 40.1 | 47.9 | 34.1 | 55.7 | 36.1 | 18.6 | 37.1 | 46.3 |
| + ODConv [Li et al., 2022] | 37.2 | 59.4 | 39.9 | 22.6 | 40.0 | 48.0 | 34.5 | 56.4 | 36.3 | 19.3 | 37.3 | 46.8 |
| + KW [Li and Yao, 2024] | 36.4 | 58.3 | 39.2 | 22.0 | 39.6 | 47.0 | 33.7 | 55.1 | 35.7 | 18.9 | 36.7 | 45.6 |
| + **KDNA** | **37.3** | **59.8** | **40.5** | **23.0** | **40.7** | **48.5** | **34.8** | **56.5** | **36.8** | **19.5** | **37.8** | **47.2** |
| ConvNeXt-Tiny | 43.4 | 65.8 | 47.7 | 27.6 | 46.8 | 55.9 | 39.7 | 62.6 | 42.4 | 23.1 | 43.1 | 53.7 |
| + KW [Li and Yao, 2024] | 44.8 | 67.7 | 48.9 | 29.8 | 48.3 | 57.3 | 40.6 | 64.4 | 43.4 | 24.7 | 44.1 | 54.8 |
| + FDConv [Chen et al., 2025] | 45.2 | 67.9 | 48.8 | 29.9 | 48.7 | 57.6 | 40.8 | 64.8 | 43.6 | 24.8 | 44.5 | 54.9 |
| + **KDNA** | **45.4** | **68.1** | **49.4** | **30.4** | **48.9** | **58.2** | **41.2** | **65.3** | **43.8** | **24.9** | **44.7** | **55.4** |

Table 3: Same-depth controls (ImageNet-1K). `Expand`: independent kernels; `Copy`: shared, no adapter; `Adapter`: KernelDNA. Best Top-1 per backbone in bold.

| Variant | Pm (M) | TP-GPU | Top-1 (%) |
|---|---|---|---|
| ResNet18 (original) | 11.69 | 8463 | 70.25 |
| Expand | 14.64 | 7744 | 73.41 |
| Copy | 8.74 | 7735 | 69.85 |
| **Adapter (ours)** | 9.24 | 7683 | **74.23** |
| ResNet50 (original) | 25.56 | 2432 | 78.44 |
| Expand | 26.67 | 2358 | 80.23 |
| Copy | 22.55 | 2355 | 75.21 |
| **Adapter (ours)** | 22.78 | 2346 | **80.95** |

Table 4: Transformer backbones (ImageNet-1K, single runs; P/C: parents/children). KW Top-1 on DeiT from Li and Yao [2024]; its params/fps from our reconstruction. Bold: best of KW/KDNA per group.

| Model / variant | Pm (M) | TP-GPU | Top-1 (%) |
|---|---|---|---|
| MobileViT-S (9 blocks) | 5.579 | 1812 | 76.86 |
| + KW ($b=1$) | 5.743 | 809 | 77.94 |
| + KDNA (9 blk, 3P/6C) | **3.741** | **1794** | 77.13 |
| + KDNA (9 blk, 4P/5C) | 4.016 | 1784 | 77.97 |
| + KDNA (18 blk, 9P/9C) | 5.796 | 1357 | **78.81** |
| DeiT-Tiny + KW ($b=1$) | 6.45 | 2316 | 73.56 |
| DeiT-Tiny + KDNA | **5.99** | **3002** | **75.11** |
| DeiT-Small + KW ($b=1$) | 24.34 | 1122 | 80.63 |
| DeiT-Small + KDNA | **11.90** | **2429** | **80.75** |

$+0.27$ Top-1, $99\%$ of baseline throughput; the shared transformer subnetwork shrinks $2.74\times$, from $2.894$ to $1.056$ M). On DeiT-Tiny, KernelDNA is $+1.55$ Top-1 and $1.3\times$ faster than KW at $7\%$ fewer parameters; on DeiT-Small it matches KW ($+0.12$) with $51\%$ fewer parameters and $2.2\times$ the throughput. These single runs (the 150-epoch MobileViT baseline, $76.86\%$, is below the 300-epoch published number) show that the factorization transfers; they are not a statistical-superiority claim.

### 5.4 Ablation Studies

**(A) Same-depth controls: where do the gains come from?** We compare three same-depth variants on ResNet18/50 (Table 3): `Expand` (independent kernels at the added positions), `Copy` (parents shared without any adapter), and `Adapter` (KernelDNA). `Copy` *degrades* accuracy below the baseline ($-0.40$/$-3.23$): an unmodulated parent cannot serve $L$ layer roles, so sharing alone is harmful. `Adapter` surpasses the parameter-rich `Expand` on ResNet18 ($74.23\%$ vs. $73.41\%$ with $63\%$ of its parameters) and ResNet50 ($80.95\%$ vs. $80.23\%$, $85\%$), so the gain is not explained by added depth or capacity; the channel gate alone on `Copy` gives $71.82\%$ (Table 8), so gating alone is not sufficient either. The modest `Adapter`–`Expand` gaps ($+0.82$/$+0.72$) indicate that inexpensive modulation recovers most of the useful layer-specific freedom of independent kernels; the result is the trade-off, not a large margin.

**(B) Parent–child placement.** Among all reasonable layouts of $\{\mathtt{F}, \mathtt{S}\}$ within a stage, `FS-SF-SF` (two parents, three children) gives the best accuracy/parameter trade-off on ResNet18 (Table 7); the three sharing layouts span only $0.81$ Top-1 points while spanning $6.29$–$9.24\,\mathrm{M}$ parameters and all exceed the baseline by $> 3$ points, so several layouts are viable (they jointly vary placement and capacity, so placement is not isolated).

**(C) Adapter components.** We ablate the three adapter components (channel/spatial/filter) on ResNet18 in Table 8. Channel attention is the most important single contributor ($+1.97\%$ alone), confirming that input-adaptivity matters most along the channel dimension; spatial and filter modulators contribute additively, and all three together give the best result. The SE reduction ratio is insensitive: $r \in \{1, \ldots, 16\}$ stays within $0.52$ Top-1 points (Table 10).

**(D) Parameter scaling and (E) training cost.** Doubling/quadrupling parents and children continues to improve accuracy ($+1.18\%/+2.35\%$ on ResNet18) without re-engineering, and training cost is $1.32\times$ the baseline—vs. $2.89\times$ for ODConv and $2.27\times$ for KernelWarehouse—because no batch-expanded weight tensors or grouped convolutions appear in backpropagation (Table 9).

## 6 Discussion and Conclusion

**Why does it help?** The same-depth controls rule out depth, capacity, sharing alone, and gating alone, leaving adapter-mediated specialization of a shared parent as the operative ingredient. Three mechanisms are plausible contributors that our experiments do not isolate: *(i)* a regularization effect of sharing (Theorem 2 quantifies the reduction in worst-case norm budget, not a generalization bound); *(ii)* better-conditioned optimization—zero-initialized modulators start every child at its parent, and the static degrees of freedom are per-layer parameters rather than learned jointly with $N$ kernels and four attentions as in ODConv; *(iii)* gradient averaging at the parent (Equation (5), a heuristic).

**What "efficient" means here.** KernelDNA's FLOPs overhead is at most $\sim 25\%$ (ResNet18; $\leq 10\%$ elsewhere), translating to a 1–$9\%$ GPU throughput drop and a 2–$28\%$ CPU latency increase over the static baselines (vs. a $6.7\times$ GPU slowdown for ODConv on ResNet18), and to $1.4$–$2.5\times$ lower CPU latency than ODConv/KW on ResNet18/50. The supported claim is *near-static throughput within the dynamic-convolution family* plus parameter savings; KernelDNA is not faster than the original static CNN. After fusion the deployed model is standard convolutions plus channel gates, which establishes operator compatibility with mobile inference engines, not measured on-device latency.

**Limitations.** (1) *Theory.* Theorem 1 is a static existence result that does not cover the trained, input-dependent gate; Theorem 2 compares worst-case norm budgets of a norm-constrained subclass and is not a generalization bound; the SNR analysis is a heuristic with a common-mean assumption. (2) *Placement* is hand-designed and architecture-dependent; the rule in Appendix B is a reproducible default, not a validated optimum. (3) *Storage vs. latency* are two deployment modes of one model: the reported parameter counts (shared representation) and throughput (fused kernels) are not attained simultaneously by one exported artifact, and the storage-optimal mode adds a batch-independent kernel-sized product per child that we have not benchmarked. (4) *Hardware.* We measured an RTX 4090 and a single-thread EPYC CPU; ARM/NPU latency and quantization/pruning compatibility are unverified. (5) *Statistics.* Multi-seed results exist only for KernelDNA on two backbones and the strongest baselines are single runs, so the smallest gaps are not statistically established; transformer results are single runs under a shortened recipe. (6) KernelDNA is a stage-level technique and benefits little when stages are very short.

**Conclusion.** KernelDNA lifts reparameterization from the layer to the stage: one parent kernel per cluster of same-shape layers, specialized by static modulators and gated by an input-dependent channel factor that linearity keeps out of the weight. The result is dynamic, parameter-efficient, executed entirely with standard operators, and yields consistent accuracy–parameter–throughput gains on CNNs with a direct transfer to transformer projections.

## Acknowledgments and Disclosure of Funding

This work was supported in part by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China under Grant JYB2025XDXM504, and in part by the National Natural Science Foundation of China under Grants 62302381 and 52441602. The authors are with the National Key Laboratory of Human-Machine Hybrid Augmented Intelligence and the Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an, Shaanxi, China.

## References

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *International Conference on Learning Representations*, 2025.

Peter L Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

Alexandre Boulch. Sharesnet: reducing residual network parameter number by sharing weights. *arXiv preprint arXiv:1702.08782*, 2017.

Kai Chen et al. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

Linwei Chen, Lin Gu, Liang Li, Chenggang Yan, and Ying Fu. Frequency dynamic convolution for dense image prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 30178–30188, 2025.

Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11030–11039, 2020.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 764–773, 2017.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27, 2014.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020.

Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-independent sample complexity of neural networks. In *Conference on Learning Theory*, pages 297–299. PMLR, 2018.

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*, pages 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

Shwai He, Chenbo Jiang, Daize Dong, and Liang Ding. Sd-conv: Towards the parameter-efficiency of dynamic convolution. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6454–6463, 2023.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529, 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

Chao Li and Anbang Yao. Kernelwarehouse: Rethinking the design of dynamic convolution. In *International Conference on Machine Learning*, 2024.

Chao Li, Aojun Zhou, and Anbang Yao. Omni-dimensional dynamic convolution. In *International Conference on Learning Representations*, 2022.

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

Yunsheng Li, Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Ye Yu, Lu Yuan, Zicheng Liu, Mei Chen, and Nuno Vasconcelos. Revisiting dynamic convolution via matrix decomposition. In *International Conference on Learning Representations*, 2021.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

Xudong Lin, Lin Ma, Wei Liu, and Shih-Fu Chang. Context-gated convolution. In *European conference on computer vision*, pages 701–718. Springer, 2020.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations*, 2022.

Bryan A Plummer, Nikoli Dryden, Julius Frost, Torsten Hoefler, and Kate Saenko. Neural parameter allocation search. In *International Conference on Learning Representations*, 2022.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Pedro Savarese and Michael Maire. Learning implicitly recurrent CNNs through parameter sharing. In *International Conference on Learning Representations*, 2019.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. Eca-net: Efficient channel attention for deep convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11534–11542, 2020.

Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

Yuwen Xiong, Zhiqi Li, Yuntao Chen, Feng Wang, Xizhou Zhu, Jiapeng Luo, Wenhai Wang, Tong Lu, Hongsheng Li, Yu Qiao, et al. Efficient deformable convnets: Rethinking dynamic and sparse operator for vision applications. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5652–5661, 2024.

Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. *Advances in neural information processing systems*, 32, 2019.

Jinnian Zhang, Houwen Peng, Kan Wu, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Minivit: Compressing vision transformers with weight multiplexing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12145–12154, 2022.

Jingkai Zhou, Varun Jampani, Zhixiong Pi, Qiong Liu, and Ming-Hsuan Yang. Decoupled dynamic filter networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6647–6656, 2021.

Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9308–9316, 2019.

# A Side-by-Side Algorithmic Comparison

Algorithm 1 contrasts the forward pass of one convolutional layer under purely-dynamic attention (ODConv) and KernelDNA's decoupled adapter, with explicit tensor shapes that expose the two bottlenecks of ODConv—a batch-expanded weight tensor and a batch-grouped convolution—and show how KernelDNA reduces to a standard `F.conv2d` on a parent kernel that is shared across multiple child layers within the stage.

# B Implementation Details

## B.1 Parent–child layouts per backbone

We instantiate the `FS-SF-SF` pattern (or its analog) within each stage of every backbone we study. Concretely:

**ResNet18.** Stages 3 and 4 (`layer3`, `layer4`) contain two BasicBlocks, each with two $3 \times 3$ convolutions. We replace the four convolutions with the layout $\mathtt{F}_1$–$\mathtt{S}_a$–$\mathtt{S}_b$–$\mathtt{F}_2$–$\mathtt{S}_c$–$\mathtt{F}_3$, where $\mathtt{F}_1, \mathtt{F}_2, \mathtt{F}_3$ are full kernels (the original convolutions are kept and re-used; $\mathtt{F}_1$ is the stride-changing entry convolution and has no children), and $\mathtt{S}_a, \mathtt{S}_b$ derive their kernels from the parent $\mathtt{F}_2$, while $\mathtt{S}_c$ derives from the parent $\mathtt{F}_3$. This gives 2 parents and 3 children per stage. The exact code is in `models/KDNA_ResNet18.py`; the per-stage parameter accounting is summarized in Table 7.

**ResNet50.** The bottleneck $1 \times 1$–$3 \times 3$–$1 \times 1$ blocks of stage 3 and 4 are reorganized so that only the $3 \times 3$ convolution is shared (the $1 \times 1$ convolutions perform channel projection and have low CKA correlation across blocks). Stage 3 uses an `F-3(SF)` layout (one parent, three children sharing it); stage 4 uses an `F-nS-F` layout. See `models/KDNA_ResNet50.py`.

**MobileNetV2.** The inverted bottleneck blocks in stages 4–7 are reorganized into `ShareClusters`. We share only the pointwise ($1 \times 1$) PW1/PW2 weights and keep depthwise convolutions independent (depthwise kernels have only $c \cdot k^2$ parameters, so sharing them yields negligible compression). The four `ShareClusters` in MobileNetV2 ($1.0\times$) have 1, 2, 1, 1 shared children respectively. Spatial attention is omitted (PW kernels have $k = 1$).

**Algorithm 1** Forward pass of one stage layer: ODConv (purely-dynamic attention) vs. KernelDNA (decoupled attention), PyTorch-like pseudocode.

```
# Input x: [B, C_in, H, W], Output y: [B, C_out, H, W]

# ============ Purely dynamic attention (ODConv): per-layer pool of N kernels ============
weight = nn.Parameter(torch.randn(N, C_out, C_in, k, k)) # N-kernel pool, per layer
kernel_attn, filter_attn, channel_attn, spatial_attn = MultiDimensionalAttention(x)
    # 4 attentions, all DYNAMIC
    # kernel_attn : [B, N, 1, 1, 1, 1] filter_attn : [B, C_out, 1, 1]
    # spatial_attn: [B, 1, 1, 1, k, k] channel_attn: [B, C_in, 1, 1]

# Batch-expanded weight tensor (memory intensive)
weight = weight.unsqueeze(dim=0) * kernel_attn * spatial_attn # [B, N, C_out, C_in, k, k]
weight = torch.sum(weight, dim=1) # [B, C_out, C_in, k, k]
weight = weight.view(-1, C_in, k, k) # [B*C_out, C_in, k, k]

# Batch-grouped convolution (off the cuDNN/TensorRT fast path)
x = (x * channel_attn).view(1, B*C_in, H, W)
y = F.conv2d(x, weight, groups=B).view(B, C_out, H, W)
y = y * filter_attn # filter_attn applied AFTER conv

# ========= Decoupled attention (KernelDNA): one parent shared by n_kids children =========
weight = parent_conv.weight # [C_out, C_in, k, k], SHARED across this stage's child layers
filter_attn = nn.Parameter(torch.zeros(C_out, 1, 1, 1)) # static filter modulator
spatial_attn = nn.Parameter(torch.zeros(1, 1, k, k)) # static spatial modulator
channel_attn = SE_like_Attention(x) # [B, C_in, 1, 1], the ONLY dynamic factor

# No batch expansion: weight is batch-independent
weight = weight * (1 + filter_attn) * (1 + spatial_attn) # [C_out, C_in, k, k]

x = x * channel_attn # [B, C_in, H, W]
y = F.conv2d(x, weight) # standard conv on cuDNN fast path

# Note: n_kids child layers in this stage share the same "parent_conv.weight",
# each owning its own (filter_attn, spatial_attn, SE).
```

**ConvNeXt-Tiny.** We share the depthwise $7\times7$ convolution within each stage; the $1\times1$ MLPs are kept independent. The decoupled adapter follows the same recipe as ResNet18.

**A default placement rule for unseen architectures.** The layouts above were hand-designed. For a new architecture we recommend the following deterministic default, which is consistent with the choices above and requires no search: traverse each stage in execution order; partition consecutive operators by semantic role and exact input/output/kernel shape; keep all stride-changing, transition, and projection-shortcut operators independent; within each eligible run, form groups of at most four consecutive operators, keep one full kernel per group as the parent and derive the remaining operators of the group from it as children. The position of the parent inside its group is a free choice: our ResNet18 layout places the children *before* their parent ($\mathtt{S}_a$–$\mathtt{S}_b$–$\mathtt{F}_2$ and $\mathtt{S}_c$–$\mathtt{F}_3$), whereas the other backbones place the parent first. For transformers, QKV, output, FFN-in and FFN-out projections are separate roles and are never shared across roles. We do not claim this default is optimal; architecture-specific refinement or NAS may improve the trade-off.

### B.2 Adapter parameter shapes

For a parent kernel $\mathbf{W}\in\mathbb{R}^{c'\times c\times k\times k}$ shared by $L$ children, the static parameters are $\alpha_s\in\mathbb{R}^{L\times1\times1\times k\times k}$ (per-child, broadcast over channels) and $\alpha_f\in\mathbb{R}^{L\times c'\times1\times1\times1}$ (per-child, per-output channel). Both are initialized to zeros so that $\widetilde{\mathbf{W}}^{(l)}=\mathbf{W}$ at $t=0$.

The SE module uses two $1\times1$ convolutions with reduction ratio $r=4$ by default and an intermediate BatchNorm layer (omitted in the original SE-Net): $\beta_c(\mathbf{X})=\sigma(\mathrm{Conv}_2(\mathrm{ReLU}(\mathrm{BN}(\mathrm{Conv}_1(\mathrm{GAP}(\mathbf{X}))))))$, where $\sigma$ is the sigmoid, so $\beta_c(\mathbf{X})\in(0,1)^c$ for every input. $\mathrm{Conv}_2$ carries a bias, so a constant gate $b\mathbf{1}_c$ $(0<b<1)$ is exactly realizable by zeroing the

weights of $\mathrm{Conv}_2$ and setting its bias to $\log \frac{b}{1-b}$; this is the witness used in Theorem 1. Each child has its own SE module with $2c^2/r$ weights.

**Per-child BatchNorm.** Each child layer maintains its own BatchNorm with affine parameters $(\gamma^{(l)}, \beta^{(l)}) \in \mathbb{R}^{c'} \times \mathbb{R}^{c'}$. This is essentially *free* (BN is already standard practice in modern CNNs) yet provides an additional per-channel affine transform that complements $\alpha_f^{(l)}$ and supplies the additive shift unavailable in pure multiplicative modulation.

### B.3 Training protocols

**ResNet18/50.** SGD with momentum $0.9$, weight decay $1 \times 10^{-4}$, base learning rate $0.1$, cosine annealing, $100$ epochs, batch $256$, label smoothing $0.1$, standard ImageNet augmentations (random resized crop, horizontal flip).

**MobileNetV2.** SGD, momentum $0.9$, weight decay $4 \times 10^{-5}$, base learning rate $5 \times 10^{-2}$, cosine annealing, $100$ epochs, batch $256$.

**ConvNeXt-Tiny.** AdamW, weight decay $5 \times 10^{-2}$, base learning rate $4 \times 10^{-3}$, cosine annealing, $300$ epochs, batch $1024$, advanced augmentations (Mixup $\alpha = 0.8$, CutMix $\alpha = 1.0$, RandAugment, label smoothing $0.1$, EMA).

These protocols match those used by KernelWarehouse [Li and Yao, 2024] for direct comparability. All baseline and competitor numbers in Tables 1 and 2 are either reproduced under these settings or reported from [Li and Yao, 2024]. The transformer recipes are given in Appendix D.5.

**Seeds.** Tables 1 and 2 report seed $42$. Table 5 lists the two additional seeds ($0$ and $1$) trained for KernelDNA on ResNet18 and MobileNetV2-1.0× under the protocols above; the reported deviation is the sample standard deviation over the three runs. The baselines were not re-run with multiple seeds, so no pairwise significance test is reported.

Table 5: Seed variability of KernelDNA (ImageNet-1K Top-1, %). Mean ± sample standard deviation over three seeds.

| Model | Seed 42 | Seed 0 | Seed 1 | Mean ± std |
|---|---|---|---|---|
| ResNet18 + KDNA | 74.23 | 74.20 | 74.24 | $74.22 \pm 0.02$ |
| MobileNetV2-1.0× + KDNA | 75.51 | 75.47 | 75.50 | $75.49 \pm 0.02$ |

### B.4 Latency and throughput measurement; deployment modes

GPU throughput is measured on an Nvidia RTX 4090 at FP32, batch $128$, inputs $224 \times 224$ ($256 \times 256$ for MobileViT-S), after $200$ warmup iterations and averaged over $1{,}000$ measurement iterations. CPU latency is measured on an AMD EPYC 7H12 (single thread, batch 1, FP32), averaged over $1{,}000$ runs. For dynamic convolution methods we report numbers *after* static fusion (where applicable); KernelDNA's static modulators are fused into the parent kernel before timing (latency-optimal mode). Parameter counts, in contrast, describe the storage-optimal representation (one parent per cluster plus the small modulators and gates). In the storage-optimal mode each child performs one additional batch-independent element-wise product of kernel size ($c'ck^2$ multiplications) per forward pass before calling `F.conv2d`; we have not benchmarked this mode. We did not measure ARM or NPU hardware.

## C Detailed Proofs

### C.1 Proof of Theorem 1 (static approximation under alignment)

*Proof.* Let $\mathbf{W}, \mathbf{W}^* \neq \mathbf{0}$ with $\mathrm{ALIGN}(\mathbf{W}, \mathbf{W}^*) = \langle \mathbf{W}, \mathbf{W}^* \rangle^2 / (\|\mathbf{W}\|_F^2 \|\mathbf{W}^*\|_F^2) \geq 1 - \varepsilon$, $\varepsilon \in [0, 1]$. Define the least-squares projection coefficient $c = \langle \mathbf{W}, \mathbf{W}^* \rangle / \|\mathbf{W}\|_F^2$ and decompose

$$\mathbf{W}^* = c\,\mathbf{W} + \mathbf{W}_\perp, \qquad \langle \mathbf{W}, \mathbf{W}_\perp \rangle = 0.$$

By the Pythagorean identity, $\|\mathbf{W}^*\|_F^2 = c^2\|\mathbf{W}\|_F^2 + \|\mathbf{W}_\perp\|_F^2$, and

$$\mathrm{ALIGN}(\mathbf{W}, \mathbf{W}^*) = \frac{(c\|\mathbf{W}\|_F^2)^2}{\|\mathbf{W}\|_F^2\,\|\mathbf{W}^*\|_F^2} = \frac{c^2\|\mathbf{W}\|_F^2}{\|\mathbf{W}^*\|_F^2}.$$

Therefore

$$\|\mathbf{W}_\perp\|_F^2 = \|\mathbf{W}^*\|_F^2 - c^2\|\mathbf{W}\|_F^2 = \big(1 - \mathrm{ALIGN}(\mathbf{W}, \mathbf{W}^*)\big)\|\mathbf{W}^*\|_F^2 \;\le\; \varepsilon\,\|\mathbf{W}^*\|_F^2.$$

Fix any $b \in (0,1)$ and choose the adapter parameters $\beta_c \equiv b\,\mathbf{1}_c$, $\alpha_f = (c/b - 1)\,\mathbf{1}_{c'}$, $\alpha_s = \mathbf{0}_{k\times k}$. The constant gate is realizable by the SE module (zero the weights of its last $1\times 1$ convolution and set its bias to $\log \frac{b}{1-b}$), and $\alpha_f$ is unconstrained, so this triple belongs to the adapter family $\mathcal{A}$. Each factor is a constant tensor, so the broadcast product is

$$\widetilde{\mathbf{W}} = \mathbf{W} \odot (b\,\mathbf{1}) \odot \big(\tfrac{c}{b}\,\mathbf{1}\big) \odot \mathbf{1} = c\,\mathbf{W}.$$

Hence

$$\big\|\widetilde{\mathbf{W}} - \mathbf{W}^*\big\|_F = \|c\mathbf{W} - \mathbf{W}^*\|_F = \|\mathbf{W}_\perp\|_F = \sqrt{1 - \mathrm{ALIGN}(\mathbf{W}, \mathbf{W}^*)}\,\|\mathbf{W}^*\|_F \;\le\; \sqrt{\varepsilon}\,\|\mathbf{W}^*\|_F.$$

□

*Remark* 3. The witness is the least-squares optimal scalar multiple of the parent, so the stated error is exact for this restricted subclass, not merely an upper bound. The full adapter family contains the witness; hence, for any fixed input (and therefore any fixed gate value), the best approximation achievable by the full family is no worse. The theorem does not assert that gradient-based training reaches this or any other particular member, and it makes no statement about input-dependent targets $\mathbf{W}^*(\mathbf{X})$. No assumption is placed on the entries of $\mathbf{W}$; parents with zero entries (*e.g.* from pruning or sparsity) are covered.

### C.2 Proof of Theorem 2 (worst-case decomposition budget)

*Proof.* **Independent class.** For $\mathcal{H}_{\text{indep}}$, $B_{\text{indep}} = \sum_{l=1}^{N} \|\mathbf{W}_l\|_F \le NB_W$ whenever $\|\mathbf{W}_l\|_F \le B_W$ for all $l$, and the value $NB_W$ is attained by any choice with $\|\mathbf{W}_l\|_F = B_W$ for all $l$. Hence $\bar{B}_{\text{indep}} = NB_W$ exactly.

**Shared class.** Write $\mathbf{K}_l = \mathbf{W} \odot \Gamma^{(l)} = \mathbf{W} + \mathbf{W} \odot \delta^{(l)}$ with $\delta^{(l)} = \Gamma^{(l)} - \mathbf{1}$ for $2 \le l \le N$, where $\delta^{(l)}$ is understood after broadcasting to the kernel shape and $\sup_{\mathbf{X}} \|\delta^{(l)}(\mathbf{X})\|_\infty \le B_\theta$. For any tensors $\mathbf{A}, \mathbf{B}$ of the same shape, $\|\mathbf{A} \odot \mathbf{B}\|_F^2 = \sum_i A_i^2 B_i^2 \le \|\mathbf{B}\|_\infty^2 \sum_i A_i^2$, *i.e.* $\|\mathbf{A} \odot \mathbf{B}\|_F \le \|\mathbf{A}\|_F \|\mathbf{B}\|_\infty$. Applying this with $\mathbf{A} = \mathbf{W}$ and $\mathbf{B} = \delta^{(l)}(\mathbf{X})$ and taking the supremum over $\mathbf{X}$,

$$\sup_{\mathbf{X}} \|\mathbf{W} \odot \delta^{(l)}(\mathbf{X})\|_F \le \|\mathbf{W}\|_F\, B_\theta \le B_W B_\theta.$$

Summing the parent once and the $N-1$ perturbations,

$$B_{\text{KDNA}} = \|\mathbf{W}\|_F + \sum_{l=2}^{N} \sup_{\mathbf{X}} \|\mathbf{W} \odot \delta^{(l)}(\mathbf{X})\|_F \;\le\; B_W + (N-1)B_W B_\theta = \big(1 + (N-1)B_\theta\big)B_W.$$

The bound is attained: take $\|\mathbf{W}\|_F = B_W$ and $\delta^{(l)} \equiv B_\theta \mathbf{1}$, *i.e.* $\Gamma^{(l)} \equiv (1 + B_\theta)\mathbf{1}$, which is realizable in $\mathcal{A}$ (constant gate $b$, $\alpha_f = ((1+B_\theta)/b - 1)\mathbf{1}$, $\alpha_s = \mathbf{0}$); then $\|\mathbf{W} \odot \delta^{(l)}\|_F = B_\theta \|\mathbf{W}\|_F = B_\theta B_W$. Hence $\bar{B}_{\text{KDNA}} = \big(1 + (N-1)B_\theta\big)B_W$ exactly.

**Ratio.** Both suprema are finite, positive, and exact, so

$$\frac{\bar{B}_{\text{KDNA}}}{\bar{B}_{\text{indep}}} = \frac{1 + (N-1)B_\theta}{N},$$

and $\frac{1+(N-1)B_\theta}{N} < 1 \iff 1 + (N-1)B_\theta < N \iff B_\theta < 1$ (using $N \ge 2$). □

*Remark* 4. Theorem 2 divides two *exact* suprema; it does not divide an upper bound by another upper bound. It is a statement about the worst-case sum of Frobenius norms of the tensors appearing in the parent-plus-perturbation decomposition of a norm-constrained subclass. We use it only as a quantitative proxy for "how much effective weight sharing removes", motivated by the fact that norm-based capacity bounds [Bartlett and Mendelson, 2002, Golowich et al., 2018] grow with weight norms; we do not derive a Rademacher-complexity or generalization bound, and the budget does not count the adapter parameters themselves (SE gates, $\alpha_f$, $\alpha_s$). The hypothesis $\sup_{\mathbf{X}} \|\delta^{(l)}(\mathbf{X})\|_\infty \le B_\theta$ is an explicit uniform assumption; it is not implied by the sigmoid gate alone, because $\alpha_f$ and $\alpha_s$ are unconstrained during training.

### C.3 Gradient flow: variance / SNR heuristic

**Setup and assumptions.** The shared parent receives the aggregated gradient

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{l=1}^{L} (1+\alpha_f^{(l)}) \odot (1+\alpha_s^{(l)}) \odot \mathbf{g}^{(l)},$$

where the sum runs over the $L$ layers that use $\mathbf{W}$ (the parent layer itself, with $\alpha_f = \alpha_s = 0$, and its children), $\mathbf{g}^{(l)} = \partial \mathcal{L} / \partial \mathbf{W}_{\text{fused}}^{(l)}$ is the gradient with respect to the static child kernel $\mathbf{W}_{\text{fused}}^{(l)} = \mathbf{W} \odot (1+\alpha_f^{(l)}) \odot (1+\alpha_s^{(l)})$, and the input-dependent gate, which multiplies the input rather than the weight, enters only through $\mathbf{g}^{(l)}$. We model each $\mathbf{g}^{(l)}$ as a random vector over mini-batch sampling with a *common* mean $\mu$ and variance $\sigma^2$ (coordinate-wise), and for $l \neq l'$ let $\rho = \mathrm{Cov}[\mathbf{g}^{(l)}, \mathbf{g}^{(l')}] / \sigma^2 \in [-1, 1]$ denote the pairwise correlation, assumed equal across pairs. At initialization the static modulation equals $\mathbf{1}$ exactly (since $\alpha_s, \alpha_f$ are zero-initialized), and we work in this regime. These are modeling assumptions chosen for tractability, not properties we have verified; in particular the common-mean assumption is violated once children specialize.

**Variance and SNR.** Under the assumptions,

$$\mathbb{E}\left[\frac{\partial \mathcal{L}}{\partial \mathbf{W}}\right] = L\mu, \qquad \mathrm{Var}\left[\frac{\partial \mathcal{L}}{\partial \mathbf{W}}\right] = L\sigma^2 + L(L-1)\rho\sigma^2 = L\sigma^2(1+(L-1)\rho),$$

so the signal-to-noise ratio of the parent's gradient is

$$\mathrm{SNR}_{\mathbf{W}} = \frac{(L\mu)^2}{L\sigma^2(1+(L-1)\rho)} = \frac{L\,\mu^2/\sigma^2}{1+(L-1)\rho}.$$

Compared with an independent kernel update, $\mathrm{SNR}_{\text{indep}} = \mu^2/\sigma^2$:

- $\rho = 0$: $\mathrm{SNR}_{\mathbf{W}} = L\,\mathrm{SNR}_{\text{indep}}$ — noise averaging.
- $\rho = 1$: $\mathrm{SNR}_{\mathbf{W}} = \mathrm{SNR}_{\text{indep}}$ — equal SNR, with the parent receiving a consistent signal of magnitude $L\mu$.
- $\rho \in (0, 1)$: $\mathrm{SNR}_{\mathbf{W}} = \frac{L}{1+(L-1)\rho}\,\mathrm{SNR}_{\text{indep}} > \mathrm{SNR}_{\text{indep}}$, since $L > 1 + (L-1)\rho \iff (L-1)(1-\rho) > 0$.

Negative correlation ($\rho < 0$) would reduce the SNR below the independent case; we did not observe it (below).

**Specialization regime.** When children specialize later in training, the per-child adapter parameters $\theta^{(l)} = (\alpha_s^{(l)}, \alpha_f^{(l)}, \mathrm{SE}^{(l)})$ have gradient $\partial \mathcal{L} / \partial \theta^{(l)} = (\partial \widetilde{\mathbf{W}}^{(l)} / \partial \theta^{(l)}) \odot \mathbf{g}^{(l)}$, which depends only on this child's loss path through $\widetilde{\mathbf{W}}^{(l)}$. The intended role of the adapters is therefore to absorb child-specific disagreement; the parent still receives the aggregated gradient above, in which differing child means partially cancel. Zero initialization of $\alpha_s, \alpha_f$ means every child starts exactly at its parent, so early training follows the common-mean regime above.

**Empirical measurement.** We measured the average pairwise correlation between the three gradient contributions $\{\mathbf{g}^{(l)}\}_{l=1,2,3}$ to one parent kernel within ResNet18 stage 3 ($L = 3$: the parent layer $\mathtt{F}_2$ and its two children $\mathtt{S}_a, \mathtt{S}_b$) throughout training: $\rho$ stays $> 0.6$ over the entire trajectory. Under the model above this corresponds to a parent-gradient SNR exceeding the independent-kernel SNR by a factor $L/(1+(L-1)\rho)$, at most $3/2.2 \approx 1.4$ for $L = 3$, $\rho \geq 0.6$. This is analogous to the optimization benefit reported for layer-wise weight sharing in ALBERT [Lan et al., 2020] and Recursive Transformers [Bae et al., 2025]. We stress that the measurement supports the correlation assumption, not the common-mean assumption, and that our ablations do not isolate gradient averaging from regularization or optimization-conditioning effects.

# D Additional Experiments

## D.1 Compute-matched comparison

A natural concern is that KernelDNA's slightly higher FLOPs (*e.g.* 2.28G vs. 1.82G on ResNet18) could partially explain its accuracy advantage. Table 6 matches FLOPs across baselines. At nearly identical FLOPs, KernelDNA still exceeds KernelWarehouse by $+0.56\%$ on ResNet18 and ODConv by $+0.33\%$ on ResNet50, with $\geq 22\%$ fewer parameters and $3.4$–$4.2\times$ higher throughput.

Table 6: Compute-matched analysis (ImageNet-1K). Best value per column within each backbone group in bold (baseline excluded for ResNet50).

| Model | Params (M) | FLOPs (G) | TP-GPU (fps) | Top-1 (%) |
|---|---|---|---|---|
| ResNet18 + KW | 11.93 | **2.23** | 2272 | 73.67 |
| ResNet18 + **KDNA** | **9.24** | 2.28 | **7683** | **74.23** |
| ResNet50 baseline | 25.56 | 4.11 | 2432 | 78.44 |
| ResNet50 + ODConv | 90.67 | **4.12** | 560 | 80.62 |
| ResNet50 + **KDNA** | **22.78** | 4.33 | **2346** | **80.95** |

## D.2 Parent–child placement and adapter components

Table 7 compares all reasonable parent–child layouts within a stage for ResNet18: the `FS-SF-SF` pattern (two parents, three children, three SE gates) achieves the best accuracy/parameter trade-off. The three sharing layouts (`FS-FSF`, `FS-SSF`, `FS-SF-SF`) jointly vary placement and parameter count; they span $73.42$–$74.23$ Top-1 and $6.29$–$9.24$ M parameters, so they demonstrate several viable operating points rather than isolating placement at fixed capacity. Table 8 ablates the three adapter components (channel/spatial/filter): channel attention is by far the most important single contributor ($+1.97\%$ over sharing alone), spatial and filter modulators contribute additively, and the full combination (✓/✓/✓) gives the best result.

Table 7: Parent–child layouts (ResNet18). Best Top-1 in bold.

| Layout | Pm (M) | TP-GPU | Top-1 |
|---|---|---|---|
| `FF-FF` (orig) | 11.69 | 8463 | 70.25 |
| `FF-FFF` | 14.64 | 7944 | 73.41 |
| `FF-FF-FF` | 17.59 | 7496 | 74.07 |
| `FS-FSF` | 9.07 | 7814 | 73.95 |
| `FS-SSF` | 6.29 | 7756 | 73.42 |
| `FS-SF-SF` (ours) | 9.24 | 7683 | **74.23** |

Table 8: Adapter components (ResNet18). Best Top-1 in bold.

| Chan | Spat | Filt | Params (M) | TP-GPU | Top-1 |
|---|---|---|---|---|---|
| ✗ | ✗ | ✗ | 8.74 | 7863 | 69.85 |
| ✓ | ✗ | ✗ | 9.24 | 7726 | 71.82 |
| ✗ | ✓ | ✗ | 8.74 | 7815 | 70.90 |
| ✗ | ✗ | ✓ | 8.75 | 7839 | 70.29 |
| ✓ | ✓ | ✗ | 9.24 | 7698 | 72.68 |
| ✓ | ✗ | ✓ | 9.24 | 7699 | 71.98 |
| ✗ | ✓ | ✓ | 8.75 | 7804 | 71.01 |
| ✓ | ✓ | ✓ | 9.24 | 7683 | **74.23** |

## D.3 Parameter scaling and training cost

Table 9 (top) shows KernelDNA's parameter scaling: doubling/quadrupling the number of parents and children continues to improve accuracy without re-engineering. Table 9 (bottom) reports training cost on a single RTX 4090: KernelDNA's $1.32\times$ training-time overhead compares favorably against $2.89\times$ (ODConv) and $2.27\times$ (KernelWarehouse) because we avoid batch-expanded weight tensors and grouped convolutions during backpropagation.

## D.4 Channel reduction ratio

Table 10 reports the effect of varying the SE channel-reduction ratio on ResNet18. All settings from $r=1$ to $r=16$ lie within $0.52$ Top-1 points of each other; we use $r=4$ as the default because it is within $0.05$ points of the best setting at $14\%$ fewer parameters than $r=1$.

Table 9: (Top) Parameter scaling of KernelDNA on ImageNet-1K. (Bottom) Training cost on ResNet18 (single RTX 4090, 100 epochs, batch 256); memory/time are end-to-end peaks. Bold: lowest cost / best Top-1 among the dynamic methods.

| *Scaling (Pm in M)* Model | Pm | Top-1 (%) | Top-5 (%) |
|---|---|---|---|
| ResNet18 baseline | 11.69 | 70.25 | 89.38 |
| KDNA-R18 ($1\times/2\times/4\times$) | 9.24/18.48/45.91 | 74.23/75.41/76.58 | 92.38/92.75/93.26 |
| MbV2 ($0.5\times$) baseline | 1.97 | 64.30 | 85.21 |
| KDNA-MbV2 ($1\times/2\times/4\times$) | 1.88/3.24/4.46 | 69.32/70.82/71.36 | 89.04/89.73/90.22 |
| *Training cost (ResNet18)* Method | Time (h) | Peak Mem (GB) | Top-1 (%) |
| ResNet18 baseline | 18.2 ($1.00\times$) | 8.4 ($1.0\times$) | 70.25 |
| + ODConv [Li et al., 2022] | 52.6 ($2.89\times$) | 22.1 ($2.6\times$) | 73.97 |
| + KernelWarehouse [Li and Yao, 2024] | 41.3 ($2.27\times$) | 15.8 ($1.9\times$) | 73.67 |
| + **KDNA (ours)** | **24.1** ($1.32\times$) | **10.2** ($1.2\times$) | **74.23** |

Table 10: Sensitivity to the SE channel-reduction ratio (ResNet18 + KernelDNA). Best value per column in bold; $r=4$ is the default used throughout.

| Reduction $r$ | Params (M) | TP-GPU (fps) | Top-1 (%) | Top-5 (%) |
|---|---|---|---|---|
| 1 | 10.72 | 7670 | **74.28** | **92.42** |
| 2 | 9.73 | 7678 | 74.25 | 92.41 |
| 4 (default) | 9.24 | 7683 | 74.23 | 92.38 |
| 8 | 8.99 | 7688 | 74.16 | 92.26 |
| 16 | **8.87** | **7691** | 73.76 | 91.91 |

### D.5 Extension to transformer projections

**Factorization.** A linear projection $\mathbf{W} \in \mathbb{R}^{d' \times d}$ applied token-wise is a $1 \times 1$ convolution, so Equation (1) applies with the spatial modulator omitted: $\widetilde{\mathbf{W}}^{(l)} = \mathbf{W} \odot \beta_c^{(l)}(\mathbf{X}) \odot (1 + \alpha_f^{(l)})$ with $\alpha_f^{(l)} \in \mathbb{R}^{d'}$. The channel gate $\beta_c^{(l)}(\mathbf{X}) \in (0,1)^d$ is produced by an SE-style block applied to the token-averaged input ("sequence channel gate") and, by linearity, multiplies the input tokens rather than the weight, so each child calls a standard `F.linear` on a batch-independent weight. Within each transformer stage we share four projection roles separately—QKV, attention output, FFN-in, FFN-out—and never share across roles. Each child owns its own LayerNorm parameters, projection biases, filter modulators $\alpha_f$ (one per projection), and two gates (one before attention, one before the FFN); the children of one parent therefore add only $O(d^2/r)$ gate weights plus $O(d)$ vectors each.

**MobileViT-S.** MobileViT-S [Mehta and Rastegari, 2022] has three transformer stages with 2, 4, and 3 blocks of width 144, 192, 240. The 3P/6C configuration uses one parent per stage (layouts `FS`, `FSSS`, `FSS`); it reduces the directly shared transformer subnetwork from $2.894\,\mathrm{M}$ to $1.056\,\mathrm{M}$ parameters ($2.74\times$) and the whole model from $5.579\,\mathrm{M}$ to $3.741\,\mathrm{M}$ ($1.49\times$), with MACs essentially unchanged ($2033\,\mathrm{M}$ vs. $2034\,\mathrm{M}$). The 4P/5C configuration keeps 9 blocks with $1:2$ parent–child groups (four parents, five children); the 18-block configuration uses $1:1$ groups (nine parents, nine children). The stem, local MobileNetV2 blocks, and classifier are untouched, which is why whole-model compression is diluted relative to the component compression. All MobileViT rows in Table 4 (baseline, KernelWarehouse, and KernelDNA) are trained by us from scratch under the same 150-epoch recipe at $256\times256$ with the `ml-cvnets` code base; this recipe is shorter than the 300-epoch schedule of the published 78.4% MobileViT-S, so our retrained baseline (76.86%) is the appropriate control. KernelWarehouse has no published MobileViT variant; we extended our reconstruction of its DeiT recipe (below) to MobileViT without modifying the warehouse mechanism ($b=1$).

**DeiT.** KernelWarehouse did not release a transformer implementation, so we reconstructed its published DeiT recipe while importing its warehouse machinery unmodified; its DeiT Top-1 values

in Table 4 are quoted from Li and Yao [2024], and its parameter counts and throughput are measured on our reconstruction. KernelDNA variants are trained from scratch under the same recipe with a $1:1$ parent–child grouping for every projection role; the Tiny and Small variants differ in their number of layers and have $5.99\,\mathrm{M}$ and $11.90\,\mathrm{M}$ parameters, respectively ($7\%$ and $51\%$ fewer than the corresponding KW models). Throughput is measured on an RTX 4090 at batch $128$ and $224\times224$.

### D.6 Behavior of the dynamic gate $\beta_c(\mathbf{X})$ after training

Theorem 1 is a static result, and a natural question is whether the trained gate is in fact input-dependent or collapses to a nearly constant vector. We measured all $12$ dynamic gates of the trained 3P/6C KernelDNA-MobileViT-S checkpoint on $256$ ImageNet validation images (evenly spaced over the validation set) and decomposed the gate tensor $\beta[\text{image}, \text{channel}]$ into $\sigma_{\text{input}}$, the standard deviation across images averaged over channels (the input-driven component), and $\sigma_{\text{chan}}$, the standard deviation across channels of the image-averaged gate (the static, layer-identity component). Table 11 lists the results. Aggregated over the $12$ gates, $\sigma_{\text{input}} = 0.171$ exceeds $\sigma_{\text{chan}} = 0.130$ (ratio $1.31$), and the observed gate values span essentially the whole sigmoid range $(0, 1)$. Because the gate is a sigmoid output, $\beta_c(\mathbf{X}) \in (0, 1)$ holds uniformly over the input space by construction; what varies across inputs is where in that range the gate sits. This is an endpoint measurement on one backbone, not a training-trajectory study or a universal claim; it establishes that the trained gate is genuinely input-dependent, and we keep this evidence separate from the static theorem.

Table 11: Dynamic-gate statistics of the trained KernelDNA-MobileViT-S (3P/6C) on $256$ validation images. $C$: gated channels; mean/min/max over all (image, channel) entries; $\sigma_{\text{input}}$: std across images averaged over channels; $\sigma_{\text{chan}}$: std across channels of the image-averaged gate; ratio $= \sigma_{\text{input}}/\sigma_{\text{chan}}$. L$s$.C$k$ denotes child $k$ of stage $s$; MHA/FFN denote the gate before attention and before the FFN.

| Gate | $C$ | mean | min | max | $\sigma_{\text{input}}$ | $\sigma_{\text{chan}}$ | ratio |
|---|---|---|---|---|---|---|---|
| L3.C0 MHA | 144 | 0.558 | 0.072 | 0.856 | 0.045 | 0.061 | 0.74 |
| L3.C0 FFN | 144 | 0.610 | 0.001 | 0.994 | 0.108 | 0.146 | 0.74 |
| L4.C0 MHA | 192 | 0.454 | 0.003 | 0.989 | 0.152 | 0.115 | 1.32 |
| L4.C0 FFN | 192 | 0.523 | 0.000 | 1.000 | 0.221 | 0.115 | 1.92 |
| L4.C1 MHA | 192 | 0.453 | 0.002 | 0.995 | 0.159 | 0.135 | 1.18 |
| L4.C1 FFN | 192 | 0.503 | 0.001 | 0.999 | 0.199 | 0.143 | 1.39 |
| L4.C2 MHA | 192 | 0.497 | 0.006 | 0.977 | 0.125 | 0.142 | 0.88 |
| L4.C2 FFN | 192 | 0.459 | 0.001 | 0.999 | 0.183 | 0.166 | 1.10 |
| L5.C0 MHA | 240 | 0.296 | 0.000 | 0.999 | 0.198 | 0.134 | 1.47 |
| L5.C0 FFN | 240 | 0.474 | 0.000 | 1.000 | 0.252 | 0.142 | 1.77 |
| L5.C1 MHA | 240 | 0.282 | 0.000 | 0.998 | 0.189 | 0.118 | 1.60 |
| L5.C1 FFN | 240 | 0.411 | 0.000 | 0.999 | 0.220 | 0.142 | 1.55 |
| Aggregate (12 gates) | – | – | 0.000 | 1.000 | 0.171 | 0.130 | 1.31 |

### D.7 CKA before and after KernelDNA

Figure 4 compares the layer-wise CKA pattern of vanilla ResNet18 with that of KernelDNA-ResNet18. The block-diagonal structure is preserved: KernelDNA does not collapse the representation; it captures the underlying structure.

### D.8 Adapter visualization

Figure 5 visualizes the spatial-attention parameters $\alpha_s^{(l)}$ of the three child layers in stage 3 and the three in stage 4 of ResNet18+KernelDNA ($\mathtt{S}_a, \mathtt{S}_b$ share the parent $\mathtt{F}_2$ and $\mathtt{S}_c$ derives from $\mathtt{F}_3$ in each stage). Children of the same parent learn distinct spatial patterns (visual diversity), confirming that the adapter mechanism produces functionally different kernels rather than near-identical replicas.

### D.9 Optional dynamic spatial gating

For tasks where input-adaptivity along the spatial dimension is critical (*e.g.*, fine-grained segmentation), $\alpha_s^{(l)}$ can be made dynamic by routing it through a small SE-style spatial branch. This recovers

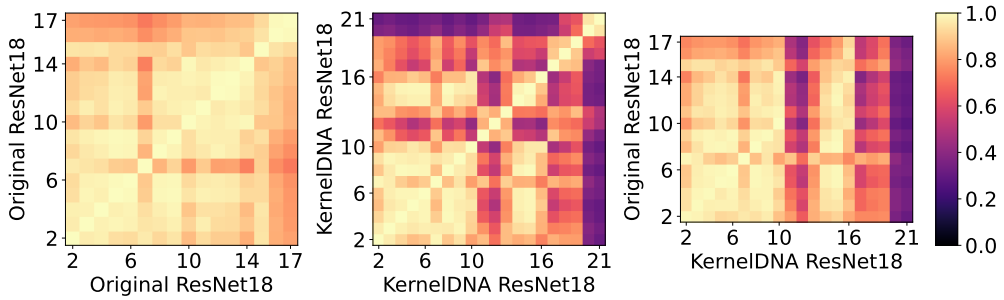


Figure 4: (Left) ResNet18 baseline CKA among $3\times3$ kernels. (Middle) KernelDNA-ResNet18 CKA. (Right) Cross-model CKA between corresponding layers. The block-diagonal structure is preserved, indicating that the adapter-modulated child kernels capture the same representational structure as independently learned kernels.

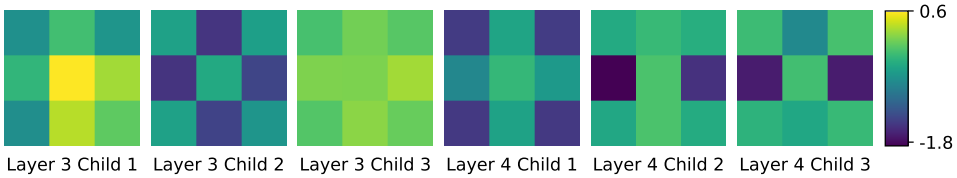


Figure 5: Visualizations of static spatial attentions $\alpha_s^{(l)}$ for child layers in ResNet18+KernelDNA. Children of the same parent learn distinct spatial patterns, showing that the adapter recovers per-layer specialization on top of a shared backbone.

ODConv-like spatial flexibility at the cost of an extra $\sim$5% latency. We treat this as an optional knob; all results in the main paper use static $\alpha_s$.

## E Reproducibility

We release the complete training code under an MIT license at `https://github.com/haiduo/KernelDNA`. The repository contains: (i) per-backbone implementations (`KDNA_ResNet18.py`, `KDNA_ResNet50.py`, `KDNA_MobileNetV2.py`) and the MobileViT/DeiT extensions; (ii) the unified training entrypoint `main.py`; (iii) measurement utilities for throughput, latency, CKA, and gate statistics; (iv) configs to reproduce every cell in Tables 1, 2 and 4. The main loop is roughly $\sim$ 600 lines of standard PyTorch. On a single RTX 4090, a 100-epoch ResNet18 run takes roughly 18 hours for the baseline and roughly a day for KernelDNA (Table 9); the larger backbones were trained on 8$\times$ RTX 4090 with the same protocol as Li and Yao [2024].

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction (Section 1) state three contributions—a cross-layer kernel-sharing framework, a decoupled static–dynamic adapter, and a theoretical analysis whose scope is stated explicitly (a static approximation guarantee and a conditional norm-budget comparison, not a generalization bound)—each backed by experiments in Section 5 and proofs in Appendix C. We state that KernelDNA is not faster than static CNNs and that its speed advantage is relative to dynamic-convolution methods.

   Guidelines:

- The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Limitations paragraph of Section 6 covers the static scope of Theorem 1, the proxy nature of Theorem 2, the heuristic status of the SNR analysis, hand-designed placement, the storage–latency deployment trade-off, the absence of on-device (ARM/NPU) benchmarks, the limited multi-seed evidence, and the dependence on stage length.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All notation is introduced before the statements (Section 4); Theorems 1 and 2 list their assumptions explicitly (nonzero kernels and an alignment condition; norm constraints and a uniform bound on the modulation perturbation). Complete proofs are in Appendix C, with sketches in the main text. The gradient-SNR analysis (Equation (5)) is labeled a heuristic with stated modeling assumptions and is not presented as a theorem.

   Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: All architectures and parent–child layouts (Section 3, Appendix B, Appendix D.5), training protocols and seeds (Appendix B), and latency/throughput measurement settings and deployment modes (Appendix B) are described; CNN recipes match those of KernelWarehouse [Li and Yao, 2024] for direct comparability.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The training code, model definitions, measurement utilities, and configurations are released under an MIT license at `https://github.com/haiduo/KernelDNA` (Appendix E). ImageNet-1K and MS-COCO are public datasets.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Optimizers, learning rates, weight decay, batch sizes, augmentations, epochs, input resolutions, and seeds are listed in Section 5, Appendix B, and Appendix D.5; the shortened (150-epoch) MobileViT recipe is stated explicitly.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: We report mean $\pm$ sample standard deviation over three training seeds for KernelDNA on two backbones (ResNet18 $74.22 \pm 0.02$, MobileNetV2-1.0$\times$ $75.49 \pm 0.02$; Table 5), where the variability captured is the training seed. The remaining results, including all baselines, are single runs following the convention of the dynamic-convolution literature, because each full ImageNet run costs roughly one RTX 4090 GPU-day for ResNet18 and far more for the larger backbones (Table 9); we therefore make no pairwise significance claims for the smallest gaps and say so in Section 5 and Section 6.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5 states the GPU/CPU hardware used for measurement. Table 9 reports training time and peak GPU memory for ResNet18 variants (baseline and all dynamic methods), and Appendix E states the hardware used to train the larger backbones.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work is a methodological contribution to convolutional and transformer architectures, evaluated on widely used public datasets (ImageNet-1K, MS-COCO). No human subjects, sensitive data, or dual-use concerns are involved.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: KernelDNA reduces the parameter count of vision models while keeping inference on standard operators, which can lower memory footprint and energy use of deployed vision systems. It does not introduce any new capability that we believe poses a misuse risk beyond existing CNN/transformer technology; general-purpose efficiency gains could marginally lower the cost of any vision application, beneficial or not.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The released artifact is a generic reparameterization technique for standard vision benchmarks (ImageNet-1K, MS-COCO) with no inherent dual-use risk; no high-risk models or scraped data are released.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All datasets (ImageNet-1K, MS-COCO), backbones (ResNet, MobileNetV2, ConvNeXt, MobileViT, DeiT), baselines (ODConv, KernelWarehouse, FDConv, and others), and toolkits (MMDetection) are cited; we use them under their respective academic/open-source licenses and terms of use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The KernelDNA codebase, model definitions, and training configurations are documented alongside the released code (Appendix E).

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: No crowdsourcing or human-subjects research was conducted.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: No human subjects were involved.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs were used only to assist with editing and proofreading; they did not contribute to the methodology, theory, experiments, or conclusions of this paper.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.