Cross-Layer Execution under Sparsity: Joint Exploration of Fusion Depth and Spatial Granularity

Hajar Falahati Adrian Cristal Osman Sabri Unsal
Barcelona Supercomputing Center (BSC)
Barcelona, Spain

Abstract

Deep neural network (DNN) accelerators often execute layers one at a time, which can require intermediate activations to be written to off-chip memory and fetched again by the next layer. Cross-layer execution can avoid part of this traffic by keeping intermediate activations on chip. Its benefit, however, depends on how many layers are processed together and how the feature maps are spatially partitioned. With sparse activations, regions with the same shape can contain different numbers of nonzeros, which changes their storage cost and the cost of overlapping receptive regions. We propose STILE, *Sparsity-aware Tiled Inter-Layer Execution*, which jointly explores the cross-layer depth L and spatial partitioning factor P using region-level activation sparsity. For each (L, P) configuration, we derive the required regions across consecutive layers and estimate peak activation storage, off-chip activation traffic, and overlap using the actual nonzero counts of these regions. We evaluate STILE using sparse activations from VGG-16 and AlexNet. The results show that sparsity changes both configuration feasibility and the selected (L, P) parameters. Across VGG-16 group-budget pairs, dense geometry-only selection, which ignores activation sparsity, and sparsity-aware selection choose different (L, P) configurations in 48.6% of cases. For VGG-16 with a 64-KiB activation-SRAM budget, sparsity-aware selection over the full candidate space reduces activation traffic by 57.0% compared with geometry-only selection and by 90.7% compared with layer-by-layer execution.

Keywords

Deep neural networks, Cross-layer execution, sparsity, Spatial tiling

1 Introduction

Deep neural network (DNN) accelerators commonly execute layers sequentially, which can require intermediate activations to be written to off-chip memory and fetched again by subsequent layers. These transfers increase off-chip traffic when intermediate activations cannot remain on chip. Prior work has explored execution strategies at different granularities to improve data locality, from intra-layer dataflows to cross-layer execution that retains intermediate data across layer boundaries.

Prior work reduces data movement at several execution granularities, as summarized in Table 1. Intra-layer dataflows such as Eyeriss [4] and SCNN [10] improve reuse within a layer, with SCNN also exploiting sparse weights and activations. Fused-layer approaches [1, 5] extend reuse across layer boundaries by retaining intermediate data and exploring tiling, retention, and recomputation trade-offs. Other approaches pipeline spatial tiles across accelerators [12] or perform operator- and compiler-level fusion [8, 9]. More recent work such as FFM [3] enables efficient optimal search over a comprehensive fused mapping space. LoopTree [5] and related frameworks [2] provide systematic mechanisms for representing and exploring fused-layer mappings. Our work is complementary to these efforts by incorporating measured region-level activation sparsity into the analysis of storage, traffic, overlap, and configuration feasibility in tiled cross-layer execution.

Cross-layer tiled execution introduces a trade-off between fusion depth, on-chip storage, and overlap among the receptive regions of neighboring output tiles. Processing more consecutive layers can eliminate additional intermediate transfers, but also enlarges the receptive regions that must be kept on chip. Finer spatial partitioning reduces the per-tile working set, but increases overlap among neighboring regions and can lead to repeated input accesses. Under sparse execution, geometry alone does not fully capture this cost. Two regions with the same shape can contain different numbers of nonzeros and therefore require different amounts of storage. The distribution of these nonzeros can also affect the peak occupancy of individual tiles. As a result, sparsity can change both which configurations are feasible and which (L, P) configuration is selected under a given SRAM budget.

To explore this design space, we propose *Sparsity-aware Tiled Inter-Layer Execution (STILE)*, which partitions the output of a group of consecutive layers into spatial tiles. For each output tile, it derives the required regions in the preceding layers and propagates only the dependent computation through the group, while keeping intermediate activations on chip. STILE parameterizes execution using the cross-layer depth L, which sets the number of consecutive layers, and the spatial partitioning factor P, which determines the output-tile granularity. For each (L, P) configuration, it estimates peak activation storage, off-chip activation traffic, and overlap overhead using the actual nonzero counts of the propagated regions.

We evaluate STILE using sparse activations from VGG-16 and AlexNet and validate the dense geometric accounting against fixed fused-layer mappings expressed in the LoopTree/AccelForge flow. At a 64-KiB activation-SRAM budget, sparsity-aware selection over the full candidate space reduces VGG-16 activation traffic by 57.0% compared with geometry-only selection and by 90.7% compared with layer-by-layer execution. available SRAM capacity.

In summary, this work makes the following contributions:

- We introduce STILE, a cross-layer tiled execution framework that jointly considers fusion depth and spatial partitioning under activation sparsity.
- We develop an analytical model that derives the required regions across consecutive layers and estimates peak activation storage, off-chip traffic, and overlap using region-level nonzero counts.
- We evaluate how SRAM capacity and activation sparsity affect cross-layer configuration selection, and how region-level nonzero distribution affects configuration feasibility.

Table 1: Comparison with representative dataflow, cross-layer, and fusion approaches.

Category	Work	Granularity	Cross-Layer Depth	Spatial/Tiling Region	Sparse	Note
Intra-layer dataflow	Eyeriss [4]	Single layer	No	Intra-layer mapping	No	Optimizes reuse and movement within each layer.
	SCNN [10]	Sparse convolution	No	Intra-layer sparse mapping	Yes	Exploits sparse weights and activations within a layer.
Fused-layer execution	Fused-Layer [1]	Adjacent Conv layers	Layer groups	Computation pyramid	No	Trades DRAM traffic against storage/recomputation.
	LoopTree [5]	Fused-layer mappings	Yes	Explicit tensor tiling	No	Jointly models tiling, retention, recomputation, and buffer/traffic costs.
Tile-grained pipeline	TGPA [12]	Tiles across accelerators	Pipeline partition	Pipeline tile	No	Optimizes mapping, latency, and utilization.
Operator/compiler fusion	DNNFusion [9]	Operators / loops	Graph-level	No explicit spatial tile	No	Selects operators and loops for fused kernels.
	FuseFlow [8]	Sparse expressions	Expression-level	No explicit spatial tile	Yes	Balances sparse movement against computation/recomputation.
This work	**STILE**	**Sparse spatial regions × layers**	**Explicit L**	**Explicit P**	**Yes**	**Models actual region-level NNZ distribution in storage, traffic, and overlap costs.**

2 Sparsity-Aware Tiled Inter-Layer Execution

Figure 1 shows the execution model of *Sparsity-aware Tiled Inter-Layer Execution (STILE)*. For a group of consecutive layers, STILE partitions the group output into $P \times P$ spatial tiles. For each output tile, it derives the required regions in the preceding layers and executes only the computation needed to produce that tile. Intermediate activations remain on chip while the tile is propagated through the selected L layers. We assume that output tiles are processed independently. Data shared by the receptive regions of different tiles is not retained across tile executions. As a result, overlapping regions may be processed more than once and overlapping portions of the group input may be fetched more than once from off-chip memory. We denote the first and last layers of a candidate group by g and $e = g + L - 1$, respectively. A_i is the output activation of layer i, so A_{g-1} is the group input and A_e is its output. For an output tile t, $R_{i,t}$ denotes the spatial region of A_i required to produce that tile.

2.1 Cross-Layer Region Construction

The group output $A_e \in \mathbb{R}^{C_e \times H_e \times W_e}$ is partitioned along its spatial dimensions into $P \times P$ non-overlapping tiles, while all channels are retained. We index a tile by $t = (t_h, t_w)$, where $t_h, t_w \in \{0, \ldots, P-1\}$, and define its boundaries as

$$\begin{aligned} h_t^s &= \left\lfloor \frac{t_h H_e}{P} \right\rfloor, \quad h_t^e = \left\lfloor \frac{(t_h+1)H_e}{P} \right\rfloor, \\ w_t^s &= \left\lfloor \frac{t_w W_e}{P} \right\rfloor, \quad w_t^e = \left\lfloor \frac{(t_w+1)W_e}{P} \right\rfloor. \end{aligned} \tag{1}$$

Thus, $R_{e,t} = [h_t^s, h_t^e) \times [w_t^s, w_t^e)$. This partitioning also supports feature-map dimensions that are not divisible by P.

Starting from $R_{e,t}$, we propagate its spatial dependency backward through the selected layers. Consider layer i, which maps A_{i-1} to A_i. For a spatial dimension $q \in \{h, w\}$, let $[x_i^{s,q}, x_i^{e,q})$ be the required interval in A_i. For kernel size k_i^q, stride s_i^q, padding p_i^q, and dilation d_i^q, the required interval in A_{i-1} is

$$\begin{aligned} x_{i-1}^{s,q} &= \max\left(0,\ x_i^{s,q} s_i^q - p_i^q\right), \\ x_{i-1}^{e,q} &= \min\left(D_{i-1}^q,\ (x_i^{e,q} - 1)s_i^q - p_i^q + d_i^q(k_i^q - 1) + 1\right), \end{aligned} \tag{2}$$

where D_{i-1}^q is the size of the corresponding spatial dimension of A_{i-1}. The clipping at the tensor boundaries excludes padded values outside the valid activation region. Applying Eq. 2 to both spatial dimensions and repeating it from layer e to g gives the regions required for tile t, $R_{e,t} \to \cdots \to R_{g-1,t}$.

For an interior region, where the boundary clipping in Eq. 2 is inactive, let $T_i^q = x_i^{e,q} - x_i^{s,q}$ denote its size in one spatial dimension. The required input-region size is

$$T_{i-1}^q = (T_i^q - 1)s_i^q + d_i^q(k_i^q - 1) + 1. \tag{3}$$

As the dependency is propagated through more layers, the required region grows. Thus, increasing L can reduce intermediate activation traffic while increasing the per-tile working set.

2.2 Sparsity-Aware Cost Analysis

Under sparse execution, a region's cost depends on the number of nonzero activations it contains, not only on its geometric size. We therefore use the actual nonzero population of each propagated region in the storage, traffic, and overlap calculations. Our analysis focuses on activation sparsity; weight storage can be modeled separately. For activation A_i and region $R_{i,t}$, we define

$$N_{i,t} = \mathrm{NNZ}\big(A_i[R_{i,t}]\big). \tag{4}$$

$\mathrm{NNZ}(\cdot)$ returns the number of nonzero elements. If b_A is the number of bytes per nonzero activation, the storage required by this region is

$$B_{i,t}^A = b_A N_{i,t} + M_{i,t}^{\mathrm{meta}}, \tag{5}$$

where $M_{i,t}^{\mathrm{meta}}$ is the metadata cost of the sparse representation. For example, a bitmap-based sparse representation requires one occupancy bit per dense activation element, in addition to the stored nonzero values. For a region containing $(D_{i,t})$ elements, its metadata cost is $(M_{i,t}^{\mathrm{meta}} = Di, t/8)$ bytes under bit-level accounting. Including this cost increases both activation storage and off-chip traffic and may change the feasible configurations. We assume zero metadata cost to isolate the effect of activation sparsity and represent an ideal sparse-storage bound. For the complete activation A_i, we similarly use $B_i^A = b_A\,\mathrm{NNZ}(A_i) + M_i^{\mathrm{meta}}$. The same representation is assumed for the layer-by-layer baseline and STILE.

2.2.1 Spatial-Overlap Overhead. Although the P^2 output tiles are non-overlapping, their receptive regions in preceding layers may overlap. Under independent tile execution, the same activation can therefore be processed by multiple tiles. We define the sparse overlap amplification as the ratio between the total number of nonzero elements processed across all tile regions and the number of unique nonzero elements covered by those regions:

$$\mathrm{Amp}_i^{\mathrm{sp}}(P) = \frac{\displaystyle\sum_{t=1}^{P^2} \mathrm{NNZ}\big(A_i[R_{i,t}]\big)}{\mathrm{NNZ}\left(A_i\left[\bigcup_{t=1}^{P^2} R_{i,t}\right]\right)}. \tag{6}$$

The numerator counts all nonzero elements processed across the tiles, including repeated elements in overlapping regions. The denominator counts the distinct nonzero activation elements covered by these regions. Thus, an amplification of one indicates no redundant sparse processing, while larger values indicate increasing

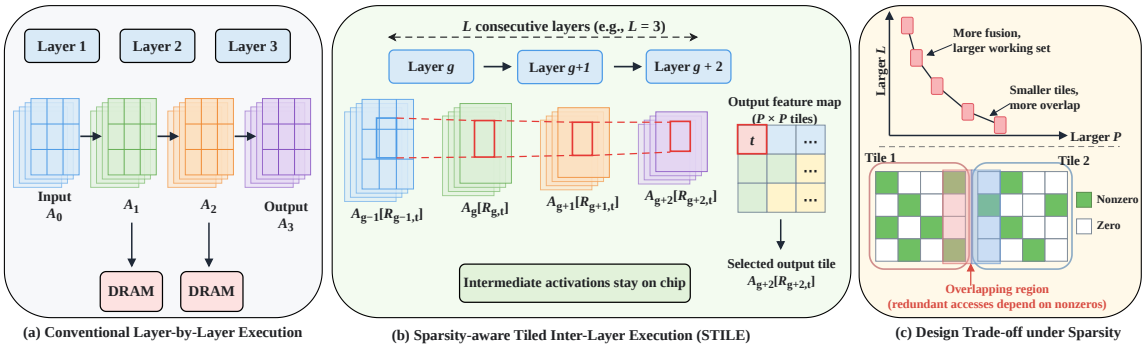


Figure 1: Overview of STILE. (a) Layer-by-layer execution materializes intermediate activations. (b) Tiled cross-layer execution for one output tile. (c) L and P trade off data movement, storage, and overlap.

overlap. For comparison, ignoring activation values and considering only region geometry, we define the corresponding dense geometric amplification as

$$\mathrm{Amp}_i^{\mathrm{dense}}(P) = \frac{\sum_{t=1}^{P^2} |R_{i,t}|}{\left| \bigcup_{t=1}^{P^2} R_{i,t} \right|}. \tag{7}$$

Here, $|R_{i,t}| = |R_{i,t}^H||R_{i,t}^W|$ is the number of spatial positions in a region. Since all channels share the same spatial region, the channel dimension appears in both the numerator and denominator and cancels in this ratio.

2.2.2 Peak On-Chip Working Set. For each tile, execution proceeds sequentially through the selected layers. Once the output region of a layer is produced, its input region can be released. Therefore, the peak activation working set is determined by the largest pair of simultaneously live input and output regions:

$$M_{\mathrm{peak}}^A(g, L, P) = \max_{\substack{t \in \{1,\dots,P^2\} \\ i \in \{g,\dots,e\}}} \left(B_{i-1,t}^A + B_{i,t}^A \right). \tag{8}$$

This value determines whether a configuration fits within the available activation SRAM. If weight storage is included, its buffer requirement can be added separately to the activation working set.

2.2.3 Off-Chip Activation Traffic. We compare STILE with layer-by-layer execution, where each intermediate activation between layers g and e is written to off-chip memory and fetched again by the next layer. The corresponding activation traffic is

$$T_{\mathrm{base}}(g, L) = B_{g-1}^A + B_e^A + 2 \sum_{i=g}^{e-1} B_i^A. \tag{9}$$

In STILE, intermediate activations remain on chip within the selected group. Off-chip traffic therefore consists of the required group-input regions and the final output tiles:

$$T_{STILE}(g, L, P) = \sum_{t=1}^{P^2} B_{g-1,t}^A + \sum_{t=1}^{P^2} B_{e,t}^A. \tag{10}$$

The final output tiles are non-overlapping, while the group-input regions may overlap and therefore be fetched more than once. Repeated processing of overlapping intermediate regions is captured separately by the overlap-amplification metric. We report the traffic reduction as

$$G_T(g, L, P) = 1 - \frac{T_{STILE}(g, L, P)}{T_{\mathrm{base}}(g, L)}. \tag{11}$$

2.3 Configuration Exploration

For a candidate group starting at layer g, a configuration is

$$C = (g, L, P), \qquad e = g + L - 1. \tag{12}$$

For each starting layer, we explore candidate depths $L \in \mathcal{L}_g$ and partitioning factors $P \in \mathcal{P}_g$. A configuration is feasible if its peak activation working set fits within the available SRAM:

$$\mathcal{F}_g = \left\{ (L, P) \in \mathcal{L}_g \times \mathcal{P}_g \;\middle|\; M_{\mathrm{peak}}^A(g, L, P) \le M_{\mathrm{SRAM}} \right\}. \tag{13}$$

Among feasible configurations, we select the one with the lowest off-chip activation traffic. Sparse overlap amplification is reported separately to quantify redundant processing due to spatial tiling. At the network level, we use dynamic programming to partition the supported layer chain into consecutive, non-overlapping groups. For each layer prefix, the algorithm considers all feasible final groups and selects the configuration (L, P) that minimizes the cumulative predicted activation traffic under the SRAM budget. Thus, fusion depth and spatial partitioning can vary across groups.

3 Evaluation

3.1 Experimental Setup

We evaluate STILE on VGG-16 [11] and AlexNet [7] using post-ReLU activations collected from 64 Imagenette [6] validation images at 224×224 resolution. The models use ImageNet-pretrained weights. We retain the original tensor shapes, layer parameters, and nonzero locations of the measured activations. The evaluated layer chains cover the feature-extraction stages of both networks: 13 convolutional and five max-pooling layers for VGG-16, and five convolutional and three max-pooling layers for AlexNet. Fully connected classification layers are excluded. Activation sparsity is measured after ReLU for convolutional layers and at the outputs of

pooling layers. We use 8-bit activation values and report activation-SRAM capacity. We assume that the weights required by each fused group are available in a separate, sufficiently large weight buffer and reused across spatial tiles. Weight-buffer capacity and weight reload traffic are not included in the reported activation costs. Therefore, our results characterize activation-data movement rather than total memory traffic or end-to-end execution performance.

Design space. For each starting layer, we explore all valid fusion depths up to the end of the supported layer chain, with $L_{\max} = 18$ for VGG-16 and $L_{\max} = 8$ for AlexNet. We use $P \in \{1, 2, 4, 8, 16\}$ for VGG-16 and $P \in \{1, 2, 4, 8\}$ for AlexNet, excluding invalid partitions. All network-level schemes use the same candidate space.

Baselines and model variants. We compare STILE with conventional layer-by-layer execution (LBL) and the exhaustive best fixed (L, P) configuration. We also consider *GEOM*, which uses dense geometric region sizes, *UNIF*, which accounts for each layer's measured sparsity but assumes it is spatially uniform, and *EXACT*, which uses the measured NNZ count of each propagated region. For each model variant, the network-level plan minimizes its predicted traffic subject to its peak-SRAM estimate. Selected plans are then evaluated using the measured sparse activations, with feasibility checked against the worst-image peak. Mapping selection is performed offline using the 64 profiled images, and the selected mapping is fixed across inputs rather than optimized separately for each image. In practice, conservative SRAM margins or a fallback to a less memory-intensive configuration could accommodate inputs exceeding the profiled peak.

Metrics. We report off-chip activation traffic, traffic reduction, and peak activation working set. Overlap amplification is used to characterize the cost of spatial tiling.

3.2 Model Validation

We validate the region-construction equations by independently enumerating the input elements required by each output tile. Across 48,100 region checks on VGG-16 and AlexNet chains, we observe no mismatches with the analytical regions. For dense geometric validation, we evaluate fixed fused-layer mappings using LoopTree [5] mapping syntax through AccelForge [2]. For representative configurations with $P \geq 2$, the unclipped formulation exactly matches AccelForge in input traffic, output writes, and peak activation storage. This agreement confirms that the dense geometric accounting of our model is consistent with the LoopTree/AccelForge evaluation flow for these mappings. In our analytical model, regions are clipped at tensor boundaries, so padded values outside the tensor are generated locally and are not stored or transferred.

3.3 Results

Joint depth-granularity trade-off. Figure 2 shows the interaction between fusion depth and spatial partitioning for the VGG-16 group starting at `conv3_1`. Increasing P reduces the peak activation working set and can make deeper groups feasible, but finer partitioning also increases overlap between neighboring receptive regions. At a 64-KiB activation-SRAM budget, $(L, P) = (4, 4)$ requires about 60 KiB and reduces activation traffic by 72.9%. Increasing the fusion depth to $L = 5$ requires a finer partition, $P = 8$, to fit within the same budget, but the traffic reduction drops to 34.8%. This example

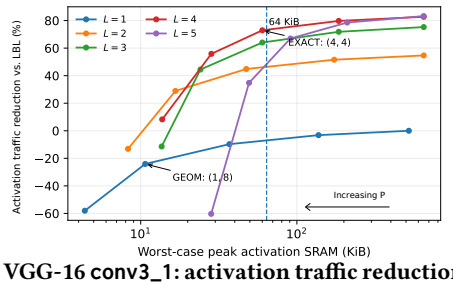


Figure 2: VGG-16 `conv3_1`: activation traffic reduction relative to the untiled layer-by-layer reference, versus worst-case peak activation SRAM.

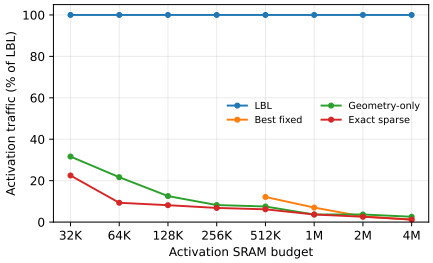


Figure 3: VGG-16 activation traffic over the full fusion-depth search space, normalized to LBL. Lower is better.

shows that deeper fusion is not always beneficial when the required spatial partitioning becomes too fine. Sparsity can also change the selected configuration itself. For `conv3_1` at 64 KiB, GEOM selects $(L, P) = (1, 8)$, whereas EXACT selects $(4, 4)$. Dense geometry estimates the latter to require about 145 KiB, while its measured sparse peak is only 60.1 KiB, making the deeper configuration feasible.

Comparison with alternative execution schemes. Figure 3 compares network-level activation traffic for VGG-16 over the full candidate space. At a 64-KiB SRAM budget, EXACT reduces traffic by 90.7% relative to LBL execution and by 57.0% relative to geometry-only selection. No single fixed (L, P) configuration is feasible for the entire network at or below 256 KiB. When a fixed configuration becomes feasible, EXACT reduces traffic relative to the best fixed (L, P) configuration by 49.2% at 512 KiB and 47.7% at 1 MiB. At 4 MiB, the best fixed and EXACT both select whole-network fusion and have the same traffic. UNIF selects the same plan as EXACT at seven of the eight SRAM budgets. At 32 KiB, however, UNIF underestimates the worst-image peak and selects a plan that requires 35.7 KiB when evaluated with the measured region-level NNZ. Thus, layer-level density captures most of the traffic effect, but region-level sparsity is needed for accurate SRAM feasibility. For AlexNet, EXACT reaches whole-network fusion $(L, P) = (8, 1)$ at 256 KiB, reducing traffic by 70.6% relative to LBL execution and by 36.9% relative to GEOM at that budget. In contrast, GEOM requires a 512-KiB SRAM budget before selecting the same whole-network configuration, $(L, P) = (8, 1)$. The effect of sparsity is not limited to changing the estimated costs; it also changes the selected configurations: GEOM and EXACT select different (L, P) configurations in 70 of 144 VGG-16 group-budget cases.

Effect of fusion depth. Figure 4 shows how network traffic changes as larger fusion depths are allowed. The useful depth depends strongly on the SRAM budget. For VGG-16 with 64 KiB of activation SRAM, increasing $L_{\max}$ from 3 to 6 reduces activation traffic from 2.562 MiB to 1.230 MiB. This corresponds to a 90.7%

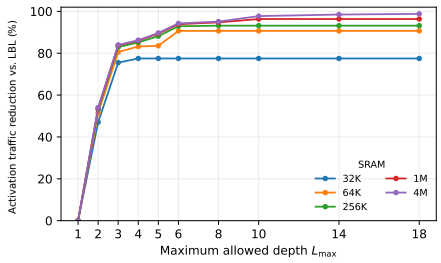


Figure 4: VGG-16 activation-traffic reduction versus the maximum allowed fusion depth $L_{\max}$ for different activation-SRAM budgets.

reduction relative to LBL. The benefit of deeper fusion does not continue at the same rate for all memory budgets. At 32 KiB, traffic saturates around $L_{\max} = 4$, while larger SRAM capacities continue to benefit from deeper groups. This shows that fusion depth should be selected together with the available on-chip capacity rather than fixed in advance.

4 Conclusion

We present STILE, a sparsity-aware approach for jointly selecting cross-layer depth and spatial granularity in tiled DNN execution. For VGG-16 with 64 KiB of activation SRAM, sparsity-aware selection over the full candidate space reduces activation traffic by 57.0% over geometry-only selection and by 90.7% over layer-by-layer execution. The results also show that the useful fusion depth changes with SRAM capacity, supporting region-level sparsity as part of cross-layer mapping decisions.

5 Acknowledgment

We would like to thank the anonymous reviewers for their valuable comments and suggestions, which helped improve the quality and clarity of this work. This work has been supported by the European High-Performance Computing Joint Undertaking (JU) under Grant Agreement No. 10120245 (DARE SGA1), by the European HiPEAC Network of Excellence, and by the Generalitat de Catalunya (contract 2021-SGR-00763).

References

[1] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-Layer CNN Accelerators. In *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. doi:10.1109/MICRO.2016.7783725

[2] Tanner Andrulis and Michael Gilbert. [n. d.]. *AccelForge*. https://github.com/Accelergy-Project/accelforge

[3] Tanner Andrulis, Michael Gilbert, Vivienne Sze, and Joel S. Emer. 2026. Fast and Fusiest: An Optimal Fusion-Aware Mapper for Accelerator Modeling and Evaluation. arXiv:2602.15166 [cs.AR] https://arxiv.org/abs/2602.15166

[4] Yu-Hsin Chen, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379. doi:10.1109/ISCA.2016.40

[5] Michael Gilbert, Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. 2024. LoopTree: Exploring the Fused-Layer Dataflow Accelerator Design Space. *IEEE Transactions on Circuits and Systems for Artificial Intelligence* 1, 1 (2024), 97–111. doi:10.1109/TCASAI.2024.3461716

[6] Jeremy Howard. 2019. *Imagenette: A Smaller Subset of 10 Easily Classified Classes from ImageNet*. https://github.com/fastai/imagenette

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*. 1097–1105.

[8] Rubens Lacouture, Nathan Zhang, Ritvik Sharma, Marco Siracusa, Fredrik Kjolstad, Kunle Olukotun, and Olivia Hsu. 2026. FuseFlow: A Fusion-Centric Compilation Framework for Sparse Deep Learning on Streaming Dataflow. In *31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 2*. 798–820. doi:10.1145/3779212.3790165

[9] Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. 2021. DNNFusion: Accelerating Deep Neural Networks Execution with Advanced Operator Fusion. In *42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*. 883–898. doi:10.1145/3453483.3454083

[10] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel S. Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks. In *44th Annual International Symposium on Computer Architecture (ISCA)*. 27–40. doi:10.1145/3079856.3080254

[11] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*.

[12] Xuechao Wei, Yun Liang, Xiuhong Li, Cody Hao Yu, Peng Zhang, and Jason Cong. 2018. TGPA: Tile-Grained Pipeline Architecture for Low Latency CNN Inference. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. doi:10.1145/3240765.3240856