Depth-First Fusion and Tiling for CNN Memory Footprint Reduction in TVM

Arthur Viens*†, Corinne Ancourt*, and Jean-Louis Dufour†
*Centre de Recherche en Informatique - Mines Paris - PSL
Fontainebleau, France
†Safran Electronics & Defense
Massy, France

***Abstract*—Deploying high-resolution Convolutional Neural Networks (CNNs) on memory-constrained embedded targets is fundamentally limited by the peak memory footprint of inter-layer activation maps, especially for high-resolution input images. While modern Deep Learning compilers employ sophisticated memory planning and buffer reuse algorithms, they typically operate within the constraints of layer-wise execution. This paradigm necessitates the materialization of full intermediate feature maps, creating unavoidable memory peaks.**

In this work, we address this bottleneck by implementing Depth-First Scheduling (DFS) within the TVM compiler stack. By bridging high-level graph partitioning (Relax) with low-level loop tiling (TIR), our approach virtualizes intermediate activations, restricting their lifetime to transient sliding windows rather than fully materialized Dynamic Random Access Memory (DRAM) allocations. Crucially, this method operates orthogonally to existing compiler optimizations, providing reductions on top of standard memory planning passes. We evaluate our framework on the MobileOne-S4 architecture with high-resolution inputs (500x500). Experimental results demonstrate a 68.8% reduction in peak activation memory compared to a strong baseline utilizing TVM's default memory planning, validating the efficacy of DFS for modern lightweight CNN architectures.

***Index Terms*—Compilation, Code optimization, Memory footprint reduction, Embedded systems, TVM, CNN.**

I. Introduction

Real world deployment of Convolutional Neural Networks (CNN) on embedded targets faces harsh resource constraints in memory footprint, energy consumption and latency requirements. Peak activation memory can become a dominant limitation especially for processing high-resolution inputs where intermediate tensors become prohibitively large.

Thus, controlling activation lifetimes and reducing the footprint of temporary buffers are becoming as crucial as minimizing arithmetic cost. However, current compilation toolchains offer limited visibility into the impact of scheduling choices on activation peaks and overall memory pressure during compilation. This lack of information regarding memory management complicates the task for developers who want to anticipate bottlenecks and generate schedules that adhere to strict memory constraints.

Existing deep learning compilers primarily focus on compute-time optimizations, few studies systematically analyze memory peaks to guide convolution optimization at the intermediate-representation (IR) level (e.g., TVM's Relax, TIR), while maintaining a hardware-agnostic representation. Consequently, compiler schedules may inadvertently generate large intermediate buffers that exceed embedded memory budgets.

At the IR level, the compiler retains access to structured information about tensor shapes, loop nesting, tile boundaries, and merge possibilities, information often lost during translation to hardware-specific code. Thus, operating at the IR level by analyzing and transforming code allows for anticipating, predicting, and reshaping memory lifetimes before actual allocation, promoting proactive memory management. This also allows transformations to remain hardware-independent while exposing enough structure to guide efficient memory planning.

This paper presents *ongoing work* towards a compile-time analysis and scheduling strategy that reduces peak memory directly at the IR level. To this end, we leverage depth-first scheduling [1]–[4] that can limit the materialization of large intermediate activations by computing convolutions over partial spatial tiles and directly consuming them in the following layers. This approach reduces the size of required live memory buffers and therefore peak memory consumption. At high level, our method (i) partitions the CNN computation graph into macro-kernels, then (ii) generates a loop-level scheduling strategy that induces tiled, sliding-window execution for each fused macro-kernel, and (iii) standard compiler passes are leveraged to minimize the storage of the remaining materialized buffers.

Our preliminary work can be summarized as follows:

- We describe a concrete implementation of depth-first scheduling implementation in TVM, leveraging loop-level schedule manipulations. We demonstrate how sliding-window buffering can be composed with standard memory reuse algorithms to minimize footprint without modifying the compiler's core allocator.
- We apply this methodology to the MobileOne-S4 CNN. Experiments on high-resolution inputs (500×500) demonstrate a 68.8% reduction in peak activation memory compared to a non-DFS baseline.

Crucially, these results highlight that the benefits of Depth-First Scheduling are orthogonal to standard memory optimization passes. Our approach achieves these reductions on top of TVM's robust default memory planning passes.

The remainder of the paper is organized as follows: Section II reviews the landscape of depth-first scheduling and memory-efficient inference, Section III details our approach before evaluating it on MobileOne-S4 in Section IV. Finally, we discuss our perspectives in Section V.

II. Related work

Conventional layer-wise (breadth-first) execution of CNNs requires materializing full activation maps in memory, which frequently exceeds the memory capacity of embedded devices. Therefore, memory-efficient execution schemes of CNNs has received significant attention, such as depth-first scheduling. DFS, also known as fused-layer or patch-based execution, avoids complete materialization by partitioning computation into spatial tiles, and immediately forwarding them through subsequent layers, thereby consuming them and reducing peak memory use.

Early experiments by Alwani et al. [1] on FPGAs demonstrated the method's effectiveness on reducing off-chip memory accesses. MicroController Units (MCUs) being highly memory constrained, MCUNetV2 [5] and StreamNet [6] showed the potential of the method on such hardware. Complementary approaches such as PEX [7] or FDT [4] explored channel-wise partial execution enabling CNN inference within a few tens of kilobytes of SRAM. These approaches demonstrate the strong potential of fine-grained memory scheduling to meet strict resource constraints. However, they remain tightly coupled to specific hardware architectures and often require manual tuning or custom runtimes. While primarily studied for inference, depth-first execution combined with checkpointing has also been shown to benefit training workloads by enabling larger image resolutions under restricted memory constraints [8]. Beyong DFS approaches, various memory-saving methods have been studied, such as tensor rematerialization exemplified in systems like Checkmate [9], or algebraic graph rewriting [10]. In parallel, modern compilers such as TVM [11], MNN [12] and others incorporate sophisticated memory planning and buffer-reuse mechanisms.

In addition to strict memory capacity constraints, embedded inference is fundamentally limited by battery life. Since off-chip DRAM accesses are orders of magnitude more energetically expensive than arithmetic operations, minimizing data movement is critical. This strategy is central to architecture-specific implementations such as MAFAT [13] and a key objective of Deeploy [14] compilation strategy for heterogeneous MCUs. However, minimizing transfers via Depth-First Scheduling is not without cost; it incurs a re-computation overhead due to overlapping halo regions in convolutional layers. As noted in StreamNet [6], aggressive fusion can result in prohibitive computational costs that negate energy savings. Consequently, efficient scheduling requires navigating the complex trade-off between arithmetic intensity and memory traffic. Pioneering works like LBDF [15] and [16] formally modeled this interaction, establishing the basis for placing the cursor between re-computation and data transfer.

While theoretical models exist, applying them across a full network creates a combinatorial space of tiling sizes, fusion depths, and compute orders that is too vast for manual tuning. To address this, recent works have integrated analytical data movement cost models into automated Design Space Exploration (DSE) engines. This approach, seen in DeFiNES [2] and Welder [17], enables systematic and rapid traversal of the scheduling space. Other frameworks employ alternative search strategies: Optimus [3] utilizes Dynamic Programming and DNNFuser [18] exploits deep learning-based estimation. Focusing specifically on memory footprint constraints, FDT [4] formulates the problem using Mixed-Integer Linear Programming (MILP). Collectively, these tools demonstrate that accurate cost estimation is a prerequisite for effective scheduling.

Despite these advances, existing methods predominantly operate at the hardware-specific or runtime level and do not provide a general mechanism for integrating memory-aware scheduling into a compiler IR. Current approaches lack a static analysis framework capable of estimating and optimizing peak activation memory during compile time. Moreover, depth-first execution strategies have not been incorporated into IR-level transformations that are portable, hardware-agnostic, and compatible with modern deep-learning compilation pipelines. The approach proposed in this work addresses these gaps by introducing a compile-time memory optimization mechanism that integrates depth-first scheduling concepts directly at the IR level, enabling deterministic memory usage estimation and automatic generation of memory-efficient execution plans.

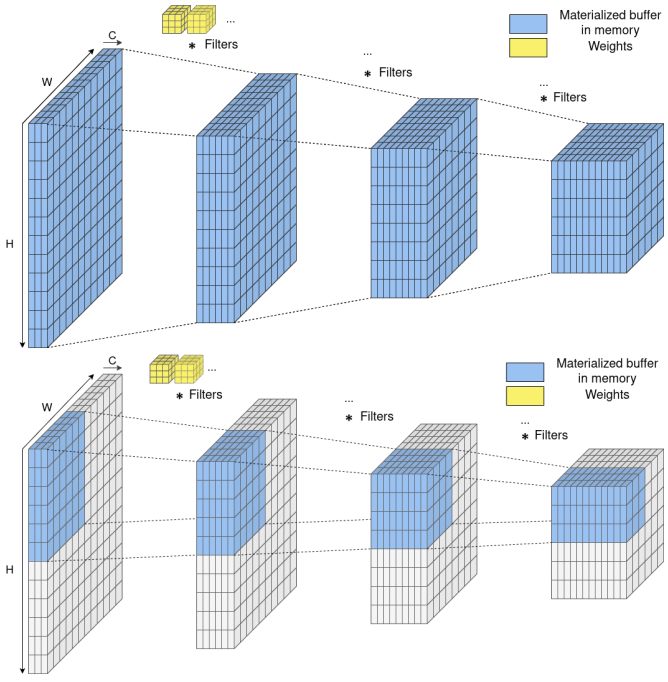

Fig. 1. Layer-wise (breadth-first) vs Depth-First CNN execution

III. Implementation of Depth-First Scheduling in TVM

We implement a Depth-First Scheduling (DFS) within the TVM compiler stack, leveraging the synergy between its two primary Intermediate Representations (IRs): Relax [19] and TIR [20]. We utilize Relax (the high-level compute graph) to manage global operator fusion and memory planning, while employing TIR (the low-level tensor IR) to explicitly manipulate loop nests, buffer scopes, and access patterns at the kernel level. Our approach fundamentally transforms the execution model from layer-wise processing to a tiled, fused pipeline. This involves three distinct stages: (1) strategic graph partitioning and kernel fusion, (2) tiling and schedule propagation, and (3) buffer minimization.

A. Graph Partitioning and Kernel Fusion

Standard TVM fusion rules typically merge element-wise operations (e.g., ReLU, Bias Add) with the preceding convolution. To enable DFS, we extend this by fusing sequences of multiple convolutional layers into "macro-kernels." However, fusing the entire network into a single macro-kernel is neither feasible nor optimal. We therefore partition the network into subgraphs driven by three constraints.

First, minimizing the *re-computation overhead*: as fusion depth increases, the overlap (halo) required for correct convolution calculation grows, eventually leading to prohibitive redundant computation that outweighs memory benefits. Second, structural limitations on the CNN model architecture impose *hard fusion barriers*, preventing fusion because of introduced dependencies in the dataflow (e.g., global pooling layers). Third, and central to our optimization on MobileOne-S4, is a partitioning heuristic driven by activation memory size. As illustrated in Fig. 2, the network exhibits significant variance in between-layer buffer sizes. We strategically align fusion boundaries (inter-kernel junctions) with the local minima of these activation sizes. This effectively encapsulates the high-footprint layers (the peaks) within the fused kernels, where they are processed as transient tiles rather than fully materialized tensors.

B. Tiling and Schedule Propagation

Once the graph is partitioned, we generate the schedule for each fused group using TVM's Tensor Intermediate Representation (TIR). A depiction of the method is available in Figure 3. Instead of allocating full activation maps, we employ a producer-consumer tiling approach. With a parameterizable user-defined tile size at the output of the fused group, we leverage TVM's analytical capabilities to back-propagate dependent tiles computation through the chain of operators (from last to first). Mechanically, this is achieved using TVM's automatic transformation primitives, which nests the computation of producer layers within the loops of consumer layers. This automatically reconstructs the producer loop nest, ensuring that producer tiles are computed immediately before they are consumed, implicitly handling the complex halo regions required by valid convolutions.

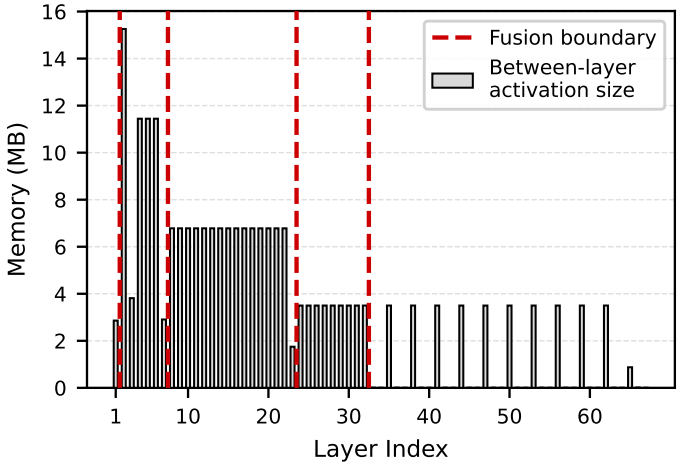

Fig. 2. Between-layer activation size in MB for MobileOne-s4 model

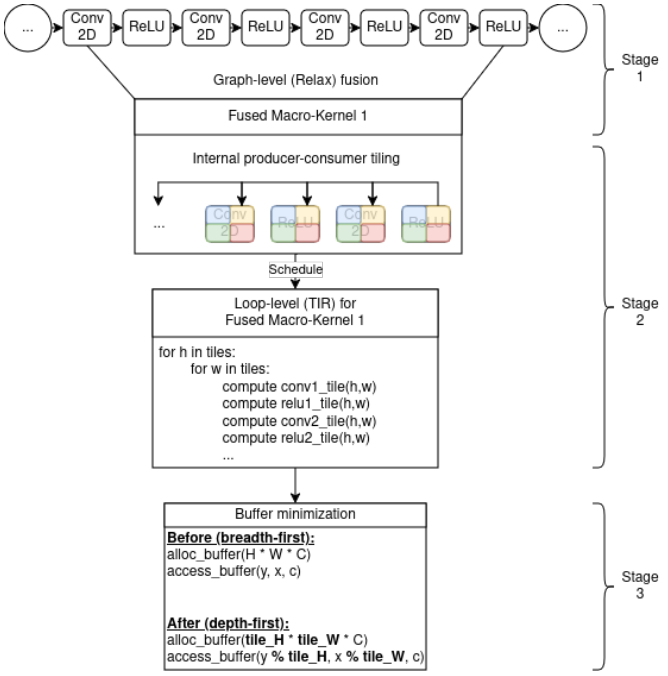

Fig. 3. Macro-kernel fusion and tiling for DFS

C. Buffer Minimization and Memory Estimation

To realize the theoretical gains of DFS, we implement a buffer minimization strategy that operates at the intra-kernel (TIR) level.

Sliding Window Addressing (TIR Level). Inside the generated TIR code, we shrink the storage buffer of intermediate feature maps to match the tile size rather than the full tensor dimensions. Since producer loops are nested within consumer loops, the compiler recognizes that only a small "sliding window" (the tile plus the necessary halo) needs to be resident in memory.

To map the original global loop indices to these shrunk buffer indices, we employ modulo arithmetic via TVM primitives. Crucially, to preserve data validity, the capacity of

these circular buffers must be rigorously lower-bounded. We determine this minimum size analytically, thereby ensuring that the producer's write pointer never overtakes the consumer's read pointer before data dependencies are resolved. This ensures that data is cyclically overwritten in a compact footprint without risking the corruption of live values.

Memory Optimization Passes. We leverage TVM's optimization passes to manage these buffers efficiently. At the intra-kernel level, TIR compiler passes analyzes the liveness of the sliding windows, reusing memory addresses for non-overlapping temporary buffers. At the graph level, Relax manages the materialized buffers, the inter-kernel junction points introduced in Section III.1, in the same manner. By combining our graph partitioning and kernel fusion strategy with tiled execution, we minimize peak memory footprint: only the carefully chosen kernel-separating activation buffers and the reduced localized tiles are active simultaneously.

IR Memory Estimation. To effectively measure the memory requirements of our execution strategy, we implemented a low-level IR parser. This tool operates prior to code generation (e.g., C, CUDA, or LLVM), analyzing the allocation and deallocation events in the IR:

- Relax Level: It tracks allocations and free events for global tensors.
- TIR Level: It tracks allocate nodes and infers deallocation from the scope of the block.

By serializing these events, we generate a precise memory timeline and calculate the peak usage, by summing current TIR kernel and Relax allocations. This allows us to account for both the reduced tile sizes and the impact of memory reuse optimizations.

IV. Experimental Results

We evaluated our DFS method on MobileOne-S4 [21], a state-of-the-art lightweight CNN designed for efficient mobile inference. Experiments were conducted using $3 \times 500 \times 500$ fp32 images, a scenario where activation memory outweights parameter storage. Consequently, this study focuses on minimizing dynamic activation memory. Weight storage optimization is reserved for future work.

Mobileone-S4 exhibits a dual structure. The initial stage consists of a strictly sequential linear branch of convolutions, on which we aggressively applied DFS, yielding substantial reductions. The latter stages are comprised of residual blocks and pooling layers with large kernel window sizes. While residual blocks do not impede the DFS applicability, pooling layers that span the complete spatial dimensions of the feature map (32x32 at this stage) collapse the dataflow dependencies. These global dependencies prevent the pipelining required for DFS. Therefore, we revert to standard layer-wise execution for these specific blocks to maintain valid execution.

As depicted in Figure 4, we compare memory consumption timelines of two executions: with and without DFS. Both approaches use identical custom fusion patterns (Conv+Bias+Activation groups of 6 to 16 operations) and default memory planning passes (StaticPlanBlockMemory + StorageRewrite) of TVM 0.21.0 version. The un-tiled variant uses TVM's default TIR lowering, which materializes complete intermediate activations for each fused operation group, resulting in layer-by-layer execution. Depth-first scheduling processes 8x8 spatial tiles through entire fused operation chains before proceeding to the next tile, enabling temporal reuse within tiles. The method does not require a specific data type, making it orthogonal to quantization, to further increase memory gains. Input images are 3x500x500 and the batch size 1. Each point represents a Relax function call with two allocated Relax tensors plus TIR kernel internals.

Ablation Study. Table I presents an ablation study isolating the impact of our DFS from TVM's memory planning and reuse passes.

1) TVM Baseline: Represents the default TVM compilation flow using standard fusion passes.
2) Custom Fusion Baseline (Un-tiled): Applies our graph partitioning (Section III.1) but executes kernels in a standard layer-by-layer fashion. Notably, this alone reduces memory compared to the TVM baseline by a factor of 32.5% (52.39 → 35.34 MB) because our inter-kernel junctions are better aligned with TVM's default memory optimization passes behaviour compared to TVM's default greedy fusion.
3) Custom Fusion + DFS: Applies our full method, enabling intra-kernel loop tiling and sliding window buffer allocation.

As shown, enabling DFS within the TIR kernels achieves **68.8% reduction** in peak memory compared to the Custom Fusion baseline, demonstrating the efficacy of tiling in constrained environments.

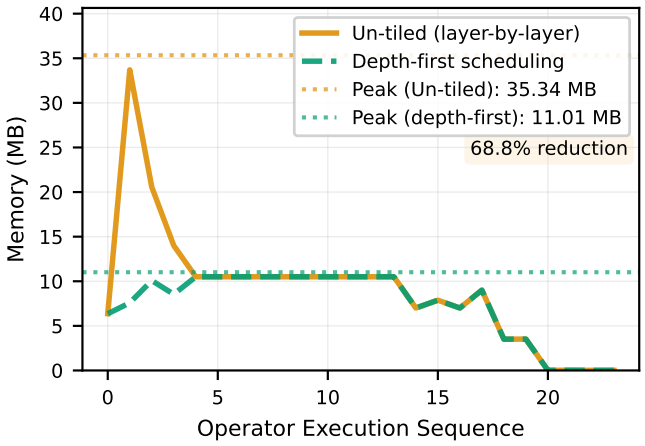

Fig. 4. Impact of depth-first scheduling on peak memory for MobileOne-S4 inference.

V. Conclusion and Future Work

This paper presented a memory-efficient compilation strategy for CNN inference on embedded targets. Our method automatically performs multi-step fusion and spatial tiling within TVM Relax-TIR pipelines, allowing the compiler to

TABLE I
ABLATION STUDY OF ACTIVATION MEMORY FOOTPRINT ON MOBILEONE-S4 ($3 \times 500 \times 500$, FP32). COMPARISON OF DEFAULT TVM HEURISTICS AGAINST OUR GRAPH PARTITIONING AND DFS TILING.

Experiment	Optimization Strategy	Peak Mem (MB)	Reduction
TVM Baseline	Default Kernel Fusion Heuristics (`FuseOps`)	52.39	(Baseline)
Custom Fusion Baseline	Graph Partitioning + Kernel Fusion	35.34	-32.5%
DFS (Ours)	**Graph Partitioning + Kernel Fusion + DFS Tiling**	**11.01**	**-79.0%**

*Reduction percentages in the table are relative to TVM Baseline.
DFS achieves a **68.8% reduction compared to the Custom Fusion Baseline.

generate optimized macro-kernels that significantly reduce peak activation memory. This mechanism preserves compatibility with standard memory planning, most scheduling heuristics, and TVM backend optimizations. Importantly, our approach operates at IR-level, remains hardware-independent, and is inherently portable to architectures supported by TVM code generation. Beyond the 68.8% memory reduction on Mobileone-S4 CNN architecture demonstrated here, this strategy provides a solid foundation for future automated search techniques: DFS can be integrated into a parameter space exploration system to explore and select optimal schedule transformation parameters, such as fusion choices and tiling sizes, under tight memory constraints.

While this work demonstrates the efficacy of Depth-First Scheduling (DFS) for activation memory reduction on MobileOne, several avenues remain to fully exploit this paradigm in constrained embedded environments.

Currently, our approach minimizes dynamic activation footprint, which is the dominant factor in high-resolution inference. However, for total SRAM compliance, static parameters (weights) must also be managed. Future iterations will integrate scheduling memory transfers for kernel weights in tandem with activation tiling to respect the on-chip memory budget. We aim to extend our partitioning and tiling method beyond the linear structures of MobileOne to support the complex DAG topologies found in state-of-the-art backbones. This entails adapting our partitioning strategy to handle nested residual connections and branching paths where managing producer-consumer dependencies become non-trivial.

Finally, as discussed in Section II, minimizing footprint may induce a penalty in memory traffic due to re-computation overhead. Consequently, a systematic exploration of the joint design space encompassing graph partitioning, tile sizes, and fusion depth remains to be conducted. Integrating an analytical cost model for DRAM traffic would allow for a fast quantitative assessment of these trade-offs, balancing strict memory constraints against latency requirements.

REFERENCES

[1] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12. [Online]. Available: https://ieeexplore.ieee.org/document/7783725/

[2] L. Mei, K. Goetschalckx, A. Symons, and M. Verhelst, "DeFiNES: Enabling Fast Exploration of the Depth-first Scheduling Space for DNN Accelerators through Analytical Modeling," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2023, pp. 570–583, iSSN: 2378-203X. [Online]. Available: https://ieeexplore.ieee.org/document/10071098/

[3] X. Cai, Y. Wang, and L. Zhang, "Optimus: An Operator Fusion Framework for Deep Neural Networks," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 1, pp. 1–26, Jan. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3520142

[4] R. Stahl, D. Mueller-Gritschneder, and U. Schlichtmann, "Fused Depthwise Tiling for Memory Optimization in TinyML Deep Neural Network Inference," Mar. 2023, arXiv:2303.17878 [cs]. [Online]. Available: http://arxiv.org/abs/2303.17878

[5] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: memory-efficient patch-based inference for tiny deep learning," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, ser. NIPS '21. Red Hook, NY, USA: Curran Associates Inc., Dec. 2021, pp. 2346–2358.

[6] H.-S. Zheng, Y.-Y. Liu, C.-F. Hsu, and T. T. Yeh, "StreamNet: Memory-Efficient Streaming Tiny Deep Learning Inference on the Microcontroller," *Advances in Neural Information Processing Systems*, vol. 36, pp. 37 160–37 172, Dec. 2023.

[7] E. Liberis and N. D. Lane, "Pex: Memory-efficient Microcontroller Deep Learning through Partial Execution," Jan. 2023, arXiv:2211.17246 [cs]. [Online]. Available: http://arxiv.org/abs/2211.17246

[8] Y. Xu, S. Raje, A. Rountev, G. Sabin, A. Sukumaran-Rajam, and P. Sadayappan, "Training of deep learning pipelines on memory-constrained GPUs via segmented fused-tiled execution," in *Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction*. Seoul South Korea: ACM, Mar. 2022, pp. 104–116. [Online]. Available: https://dl.acm.org/doi/10.1145/3497776.3517766

[9] P. Jain, A. Jain, A. Nrusimha, A. Gholami, P. Abbeel, J. Gonzalez, K. Keutzer, and I. Stoica, "Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 497–511, Mar. 2020.

[10] A. Artemev, Y. An, T. Roeder, and M. van der Wilk, "Memory safe computations with XLA compiler," *Advances in Neural Information Processing Systems*, vol. 35, pp. 18 970–18 982, Dec. 2022.

[11] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "{TVM}: An Automated {End-to-End} Optimizing Compiler for Deep Learning," 2018, pp. 578–594. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/chen

[12] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, T. Yu, C. Lyu, and Z. Wu, "MNN: A Universal and Efficient Inference Engine," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 1–13, Mar. 2020.

[13] J. Farley and A. Gerstlauer, "MAFAT: Memory-Aware Fusing and Tiling of Neural Networks for Accelerated Edge Inference," 2023, vol. 669, pp. 78–88, arXiv:2107.06960 [cs]. [Online]. Available: http://arxiv.org/abs/2107.06960

[14] M. Scherer, L. Macan, V. J. B. Jung, P. Wiese, L. Bompani, A. Burrello, F. Conti, and L. Benini, "Deeploy: Enabling Energy-Efficient Deployment of Small Language Models on Heterogeneous Microcontrollers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 4009–4020, Nov. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10745806/

[15] K. Goetschalckx and M. Verhelst, "Breaking High-Resolution CNN Bandwidth Barriers With Enhanced Depth-First Execution," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 323–331, Jun. 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8667835

[16] M. Shi, P. Houshmand, L. Mei, and M. Verhelst, "Hardware-Efficient Residual Neural Network Execution in Line-Buffer Depth-First Processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 690–700, Dec. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9570718

[17] Y. Shi, Z. Yang, J. Xue, L. Ma, Y. Xia, Z. Miao, Y. Guo, F. Yang, and L. Zhou, "Welder: Scheduling Deep Learning Memory Access via Tile-graph," 2023, pp. 701–718. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/shi

[18] S.-C. Kao, X. Huang, and T. Krishna, "DNNFuser: Generative Pre-Trained Transformer as a Generalized Mapper for Layer Fusion in DNN Accelerators," Jun. 2022, arXiv:2201.11218 [cs]. [Online]. Available: http://arxiv.org/abs/2201.11218

[19] R. Lai, J. Shao, S. Feng, S. Lyubomirsky, B. Hou, W. Lin, Z. Ye, H. Jin, Y. Jin, J. Liu, L. Jin, Y. Cai, Z. Jiang, Y. Wu, S. Park, P. Srivastava, J. Roesch, T. C. Mowry, and T. Chen, "Relax: Composable Abstractions for End-to-End Dynamic Machine Learning," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. New York, NY, USA: Association for Computing Machinery, Mar. 2025, pp. 998–1013. [Online]. Available: https://doi.org/10.1145/3676641.3716249

[20] S. Feng, B. Hou, H. Jin, W. Lin, J. Shao, R. Lai, Z. Ye, L. Zheng, C. H. Yu, Y. Yu, and T. Chen, "TensorIR: An Abstraction for Automatic Tensorized Program Optimization," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, Jan. 2023, pp. 804–817. [Online]. Available: https://dl.acm.org/doi/10.1145/3575693.3576933

[21] P. K. A. Vasu, J. Gabriel, J. Zhu, O. Tuzel, and A. Ranjan, "MobileOne: An Improved One Millisecond Mobile Backbone," 2023, pp. 7907–7917.