msf-CNN: Patch-based Multi-Stage Fusion with Convolutional Neural Networks for TinyML

Zhaolan Huang [*] and Emmanuel Baccelli[1, 2, 3]

[1]Freie Universität Berlin
[2]Einstein Center Digital Future
[3]Inria, France

Abstract

AI spans from large language models to tiny models running on microcontrollers (MCUs). Extremely memory-efficient model architectures are decisive to fit within an MCU's tiny memory budget e.g., 128kB of RAM. However, inference latency must remain small to fit real-time constraints. An approach to tackle this is *patch-based fusion*, which aims to optimize data flows across neural network layers. In this paper, we introduce *msf-CNN*, a novel technique that efficiently finds optimal fusion settings for convolutional neural networks (CNNs) by walking through the fusion solution space represented as a directed acyclic graph. Compared to previous work on CNN fusion for MCUs, msf-CNN identifies a wider set of solutions. We published an implementation of msf-CNN running on various microcontrollers (ARM Cortex-M, RISC-V, ESP32). We show that msf-CNN can achieve inference using 50% less RAM compared to the prior art (MCUNetV2 and StreamNet). We thus demonstrate how msf-CNN offers additional flexibility for system designers.

1 Introduction

Artificial Intelligence of Things (AIoT) is a domain aiming to embed AI in the smallest networked devices [11]. As such AIoT is pushing the miniaturization of Deep Neural Networks (DNNs) to fit microcontroller-based hardware, which enables various applications at the edge of the network. Use-cases include vision/audio recognition, environmental monitoring, personalized medical care, etc. However, imbalance between the increasing resource requirements of DNNs and the very limited computation capacity (CPU in MHz) and memory resource of Microcontroller Units (MCUs) remains a challenge in deploying DNNs on Internet of Things (IoT) devices. For instance, as described in RFC7228 [5], billions of IoT devices are resource-constrained devices, with Random Access Memory (RAM) smaller than 50 KiB, and Flash memory smaller than 250 KiB. On the other hand, even a single convolutional layer in quantized ResNet-34 [21, 14] consumes around 414.72 KiB in RAM. This example highlights the huge gap between memory budgets on IoT devices and RAM usage of DNNs.

A technique aimed at decreasing this gap is patch-based layer *fusion*, introduced in [2]. Initially targeting FPGAs, patch-based fusion reduces off-chip Dynamic RAM (DRAM) requirements and communication bus transfer costs for inference with CNNs. Fusion is great for low-memory devices because it can save up to 95% of RAM usage. Moreover, Fusion decouples input size from memory usage, allowing for larger input. Recent work has thus explored the use of fusion on MCUs, for example, to improve the memory consumption of the first few convolutional layers of MobileNetV2 [26].

[*]Corresponding Author. E-mail: zhaolan.huang@fu-berlin.de

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Nevertheless, we observe that significant issues linger on MCUs. First, intermediate feature maps inside the fusing block incur a high (re)compute cost. Second, input size limits hamper many use-cases such as medical image processing, sequence time series analysis (e.g. audio application), etc. Third, implementations of fusion on MCUs have so far been very hardware-specific (e.g. bound to the ARM-Cortex-M7 instruction set) and model-specific (e.g. bound to CNN mobile inverted blocks).

Contributions – With the goal of improving on the above issues, we report on following work:

- We propose msf-CNN, a fusion-based approach to achieve ultra-low RAM footprint of neural network inference and we open-source its implementation[2];
- We formulate the problem of finding optimal fusion settings that minimize peak RAM usage or compute cost of neural networks as a variant shortest path problem.
- We provide graph models representing multi-stage fusion neural networks, which encode peak RAM usage and compute cost of single and fused layers.
- We designed a pruning strategy to squeeze the search space and use graph-based algorithm to find solutions in reasonable time complexity (from $O(2^{N-2})$ to polynomial time).
- We improved global pooling and dense operators to further squeeze RAM usage without compute overhead.
- We released preliminary evaluation results on MCU-based IoT boards. We compared common CNN, StreamNet, MCUNetV2 and msf-CNN on a variety of microcontrollers. We show that msf-CNN allows new trade-off between memory saving and compute overhead.

While our main focus is on microcontrollers, msf-CNN is not limited to them. Its cost estimator and C-code backend also support general CPU platforms (e.g., x86, Cortex-A), enabling broader use such as memory-optimized cloud inference. Moreover, with appropriate cost models and backends, msf-CNN can be extended to accelerators (GPUs, FPGAs, ASICs). We leave non-MCU experiments to future work to maintain focus on devices with tighter RAM constraints.

2 Background

Patch-based Fusion for DNN on FPGA & GPU – Patch-based fusion was initially proposed in [2] as a fusion scheme for Convolutional Neural Network (CNN) deployed on Field Programmable Gate Array (FPGA) to reduce the off-chip DRAM usage and I/O overhead. Instead of computing the complete feature maps for each layer, it fuses convolutional layers into a single block (pyramid structure) and computes only one or a few output elements. This approach requires only small portions (tiles) of the feature maps loaded onto DRAM. However, the reduction of RAM is at the cost of re-computing the overlapped elements in feature maps required by adjacent fused layers. DeFiNES [28], another fusion framework, explored different cache strategies within fused layers to alleviate the re-computation issue. (Fully-recompute, H-Cached & V-recompute, and Fully-cache). Fully-recompute eliminates caching entirely, requiring all overlapping input tensor elements to be recalculated; H-cached & V-recompute caches elements along the horizontal axis while recomputing vertical overlaps; and Fully-cache retains all overlapping elements in memory. These approaches illustrate a critical trade-off—enhanced caching progressively reduces compute redundancy but proportionally increases RAM usage, with cached element quantity inversely correlating to compute overhead and directly scaling with memory demands. Additional work has also applied fusion on GPUs, for instance [31] used it for cancer detection in medical pictures. Note that *patch-based* fusion is fundamentally different from *kernel* fusion techniques. We elaborate on this distinction and its implications in Appendix A.

Patch-based Fusion on MCUs – Work on MCUNetV2 [26] has applied fusion on MobileNetV2 to reduce the peak RAM usage. It revealed that layers at the head of the model dominate the RAM usage. Hence these layers were fused into one block to reduce RAM usage significantly. The recompute issue was mitigated by redistributing the receptive field, so the receptive field inside the fusion block was decreased and regained at a later stage. Work on StreamNet [46] introduced a two-dimensional tensor cache to significantly reduce re-compute operations in a fusion block and applied brute force to search for optimal fusion position and cache depth. Nevertheless, no prior work explored the potential of multiple fusion blocks in CNNs.

[2]Please check `https://github.com/TinyPART/msf-CNN`

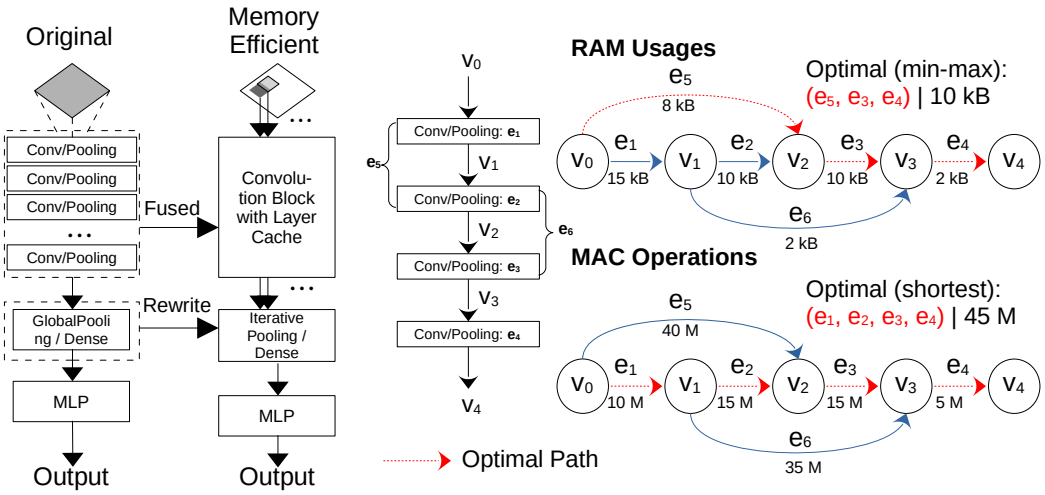
Representing DNNs as Inverted Dataflow Graphs – Dataflow graph have been widely used for modeling DNN, as pioneered by TensorFlow and PyTorch [1, 30]. The data (tensor) flows alongside the directed edge between nodes which indicates the operations (convolution, pooling, addition, etc.) applied on the incoming edges (tensors). This representation shows the producer-consumer relations among operations and has great expressiveness and flexibility, enabling automatic differentiation and concurrent execution of independent operations.

3 High-Level Idea

Inspired by the above previous works, msf-CNN aims to answer the following questions: **(1)** Where to fuse and how to determine the fusion position/depth? **(2)** Under specific resource constraints, how to find the optimal fusion settings?

As depicted in Figure 1a, msf-CNN determines fusion settings (fusion position and depth), transforms layers accordingly into fusion blocks and rewrites global pooling and dense layers as their iterative implementation, which can further squeeze RAM usage without any computation overhead.

To guide us in doing so, we use *inverted* dataflow graphs to model CNNs, where tensors are represented as nodes, and operations are depicted as edges connecting them. On this graph, we encode into the edges the resource usage of the operations, and use additional edges to represent fusion blocks. This allows us to design graph-based strategies to find optimal solutions with lower computational complexity using proven graph algorithms.



(a) Overview (b) Neural Network as DAG

Figure 1: **(a)** Overview of msf-CNN. The convolutional layers are fused into several fusion blocks based on the optimal setting found by optimizer. We let global Pooling and dense layers compute the outputs iteratively to further squeeze RAM usage. **(b)** The neural network is modeled as a directed acyclic graph (DAG). Nodes v_n denote the tensors that are produced and consumed by the operators or possible fusion blocks. Edges $e_1, \ldots, e_4$ represent individual operators, while edge e_5, e_6 represent two candidate fusion blocks. Edges are annotated with the RAM usage and multiply–accumulate (MAC) amounts of their corresponding operators and fusion blocks.

4 Problem Definitions & Assumptions

We aim to solve a pair of dual optimization problems. Let χ be the set of all possible configurations for fusion blocks. We **define *P1* as the problem of minimizing peak RAM usage** subject to a computation cost limit:

$$\min_{S} P(\chi, S) \tag{1}$$

$$\text{s.t. } F(\chi, S) < F_{max} \tag{2}$$

where P is the peak RAM usage, and F is the computation overhead for inference under fusion setting S, relatively to inference without fusion (thereafter denoted *vanilla*). The compute cost limit and RAM limit are annotated by F_{max} and P_{max}, respectively. Dually, we **define *P2* as the problem of minimizing computation cost** subject to a RAM footprint limit:

$$\min_{S} F(\chi, S) \tag{3}$$

$$\text{s.t. } P(\chi, S) < P_{max} \tag{4}$$

Without loss of generality, we only discuss fusion blocks of convolutions. We assume a *H-Cache* scheme, which we chose to be a good trade-off between buffer size and recompute cost on MCUs.

In Appendix B and Appendix C, we further detail the analysis of the cache buffer size and the amount of MAC operations.

5 DNN Graph Representation & Formulation

We interpret the optimization problems described in Section 4 by modeling the DNN as data-nodes graph. We transform the problem as a shortest path problem [35] and use off-the-shelf graph algorithms to find a solution that minimizes the peak memory usage as well as compute cost during inference regarding specified constraints.

5.1 DNN Representation

As described in Section 2, we model a DNN as a DAG $G = (V, E)$ with data nodes $v_0, \ldots, v_n$ representing input/output tensors of consecutive layers and m edges $e_1, \ldots, e_m$ that represent single layers or fusion blocks. Each edge is also encoded with resource requirements by layer or fusion block. Specifically, the first (v_0) and the last node (v_n) are the input and output tensor of the neural network, respectively.

In general, the edge represents the input/output relation of nodes and also indicates the fusion depth inside the neural network. For example, an edge that connects consecutive vertices $e = v_n \rightarrow v_{n+1}$ is a single layer that consumes v_n as input tensor and outputs tensor v_{n+1}, while an edge that jumps over multiple vertices $e = v_n \rightarrow v_{n+m}, m > 1$ represents a fusion block with m layers. Each **complete compute path** from v_0 to v_n represents a fusion setting S.

A typical example depicted in Figure 1b explains how to use DAG for representing a simple neural network. Tensors are transformed into nodes, operators and fusion blocks are edges. Edges are encoded with RAM usages and MAC amounts of their corresponding operators. Hence, the problem is transformed to find an optimal path from the input node to the output node of the graph.

5.2 Encoding RAM Usage

We first calculate the RAM usages P_{e_i} of all single layers and all possible fusion blocks inside the neural network by

$$P_{e_i} = I + O + Buf \tag{5}$$

where I and O are the size of input and output tensor, respectively. Buf represents the cache buffer size of the fusion block, which is determined by the chosen cache scheme. In this work it is given in Equation (11). Trivially, for non-fused layers Buf is always set to zero since no fusion cache is needed.

Thereafter, the calculated RAM usages are attached to the corresponding edges for further analysis. For a complete compute path contains n edges $S = (e_{i_1}, \ldots, e_{i_n})$ we can then calculate the overall peak RAM usage P_S by

$$P_S = \max_{j=1\ldots n} P_{e_{i_j}} \tag{6}$$

5.3 Encoding Compute Cost

The encoding steps of compute cost are similar to encoding peak memory usage. Here we use MAC operations as the indicator of compute cost. In this paper, the MAC amount of fusion block is given in Equation (14) and Equation (15).

After attaching the calculated MACs to the edges, the total compute cost of a complete compute path S is

$$C_S = \sum_{j=1}^{n} C_{e_{i_j}} \tag{7}$$

Therefore, the **compute overhead factor** F representing the ratio of the MAC amount after fusion to the vanilla, common one without fusion is expressed as $F = C_S/C_{vanilla}$. For the constraints in Equation (2), users can set a maximum compute overhead factor F_{max} expressed as $F_{max} = C_{max}/C_{vanilla}$. In the following sections, we will discuss several graph-based algorithms to solve the optimization problem.

6 Searching for Optimal Fusion Settings

After building an inverted dataflow graph of a DNN with all possible fusion combinations (edges), the two dual problems are indeed transformed into classic graph problems: finding an optimal complete compute path from the input tensor node v_1 to the output tensor node v_n under specific constraints.

Impact of Search Space Size – If we consider the unconstrained optimization, the solution is trivial: the single-source-single-target shortest path, which can be found by classical graph algorithm like Dijkstra's [8] with the time complexity of $\mathcal{O}(E \log(V))$. However, when considering the constraints, it is necessary first to explore all possible complete compute paths that meet the conditions, which can potentially explode the complexity to $\mathcal{O}(2^{V-2})$ [32] in the worst case. Hence, we need a smarter strategy to squeeze the search space and avoid horrendous complexity.

Note that such shortest path computations do not take place on the microcontroller at runtime. Instead, they are computed offline, on a PC, which expands the realm of what can be assessed as bearable computation.

6.1 Problem P1: Minimizing Peak RAM Usage

The unconstrained optimization is to find a complete compute path with minimal peak RAM usage, which is equivalent to finding the path that minimizes the maximum weight of edges (minimax path problem). As mentioned above, this can be solved by modified Dijkstra algorithm. An example path with minimal peak RAM usage is presented in Figure 1b.

For the constraint of compute cost limit (Equation (2)), the pruning strategy needs co-design with its optimization problem (Equation (1)). We noticed that all possible peak RAM usages have already been encoded into the edges. Therefore, the problem can be transformed into the following: we first construct a **candidate solution set** $\{S_0, S_1, \ldots, S_i, \ldots\}$ with

$$S_i = \arg\min_{S} C(G_i, S), \tag{8}$$

$$G_i := \text{subgraph of } G_{i-1}, \text{ obtained by removing all edges in } G_{i-1} \text{ with the maximal RAM usage}, \tag{9}$$

$$G_0 = G \tag{10}$$

where $C(G_i, S)$ is the MAC amount of fusion setting S in graph G_i. The candidate solution S_i can be obtained by applying the shortest path algorithm. We then filter the candidate solutions to find those that satisfy the constraints and select the one with the smallest RAM usage as the optimal solution.

In this way, we avoid constructing a search space with a complexity of $\mathcal{O}(2^{V-2})$. Instead, we iteratively eliminate subgraphs and solve for candidate solutions, reducing the complexity to $\mathcal{O}(V^3)$. For most deep neural networks running on MCUs, this process can be done in few seconds.

6.2 Problem P2: Minimizing Compute Cost

We first discuss the unconstrained variant, which is identical to $P_{max} = \infty$. In this case, finding the solution is equivalent to finding the shortest complete compute path – the path with a minimal sum of MAC – of the graph, which can be again solved by classical algorithm like Dijkstra's [8]. Figure 1b shows an example with an optimal path marked in red.

When bringing back the constraint of RAM limit, the pruning step is simple: eliminating all edges with encoded RAM usage exceeding the limit. So, all paths in the graph will automatically fulfill the limitation.

6.3 Analytical Results

To explore the capability of these two dual optimizers, here we choose three variants of MobileNetV2 and MCUNet [34, 26] with different scales for the pilot study: MobileNetV2 with width multiplier 0.35 and input size of $144 \times 144 \times 3$ (MBV2-w0.35), MCUNetV2-VVW-5fps with input size of $80 \times 80 \times 3$ (MN2-vvw5), MCUNetV2-320KB-ImageNet with input size of $176 \times 176 \times 3$ (MN2-320K). For optimizer of minimizing peak RAM usage, the maximal compute overhead factor ranges from 1.1 to 1.5 then jumps to Infinite, which represents an unconstrained optimization. For optimizer of minimizing compute cost, the maximal peak RAM usage was set from 16 kB to 256 kB where each level represents a popular RAM capacity of mainstream MCUs.

As shown in Table 1, both optimizers can indeed theoretically suppress the peak RAM usage without violating all preset constraints. The high RAM usage compression is achieved with increase of deep fusion blocks, thereby introducing a high compute overhead. The extreme cases lay on the unconstrained optimization minimizing the RAM usage by more than 90%, while reluctantly introducing $1.6\times$ to $2.7\times$ of compute overhead. This is only suitable for time-intensive applications with a high limited RAM budget.

On the other hand, setting appropriate constraints can still lead to well-optimized configurations, with our tools offering flexibility to accommodate real-life scenarios. Under different thresholds on compute overhead factor or peak RAM usage, the solutions that optimizer found are all fulfill the constraints and with RAM usage all lower than the vanilla, un-fused setting. In some cases, it is even possible to compress RAM usage without incurring additional computational overhead. These pilot studies demonstrate the effectiveness of finding usable solutions under real-life constraints.

We also conducted a preliminary analysis of the heuristic strategy used in MCUNetV2, which fuses only the early layers to minimize RAM usage. While this approach is simple and straightforward, it tends to yield suboptimal fusion configurations. As shown in Table 1, msf-CNN discovers better solutions and offers greater flexibility compared to the heuristic strategy. The analytical results were further validated by on-board experiments presented in Section 8.

Table 1: Analytical results with msf-CNN under different constraints. Vanilla: un-fused models. Heuristic: minimize RAM consumption by only fusing heading layers. SAA: Same as above. Gray: msf-CNN beats heuristic.

		MBV2-w0.35		MN2-vww5		MN2-320K	
	Constraint	*RAM (kB)*	F	*RAM (kB)*	F	*RAM (kB)*	F
Vanilla	-	194.44	1.00	96.00	1.00	309.76	1.00
Heuristic	-	32.08	1.59	24.00	1.56	90.31	3.25
P1: F_{max}	1.1	67.91	1.10	32.79	1.04	190.10	1.04
	1.2	(SAA)		26.13	1.11	186.74	1.19
	1.3	21.29	1.30	17.76	1.30	186.03	1.25
	1.4	15.34	1.38	13.38	1.35	156.67	1.37
	1.5	(SAA)		(SAA)		94.18	1.45
	Inf	7.89	1.68	12.00	1.96	42.64	2.69
P2: P_{max}	16 kB	15.34	1.38	13.38	1.35	(No Solution)	
	32 kB	25.67	1.25	26.13	1.11		
	64 kB	63.74	1.23	38.58	1.02	62.88	2.02
	128 kB	83.07	1.02	89.60	1.00	94.18	1.45
	256 kB	181.44	1.00	(SAA)		247.81	1.00

7 msf-CNN Implementation Details

We have implemented the msf-CNN fusion mechanism on top of microTVM v0.16.0 [7]. We use the TVM frontend to convert models into intermediate representation (IR), and rewrite the compute graph

and low-level routines of operators to fit the fusion settings. We used RIOT-ML [17] to benchmark the fused models: the models are transformed into C code using microTVM and integrated in dedicated firmware leveraging a common IoT operating system (RIOT [4]) which we can run on various boards shown in Table 2.

Sequential RAM Usage – We have optimized the RAM usage of the global pooling and fully connected (Dense) layers. We observed that the outputs of these two basic blocks can be computed iteratively, and in most scenarios, their input dimensions are much larger than their output dimensions. As a result, we can temporally divide the input and sequentially process it through the iterative global pooling or dense layers, which further minimizes memory usage. If their upstream is a fusion block, this perfectly matches the feature of temporally split inputs, enabling them to be fused seamlessly.

Iterative Computation of Global Pooling – As illustrated in Figure 2, standard global pooling requires that all elements of the input tensor stored in RAM. In our approach, the global pooling layer receives one or a few input elements at each step and iteratively updates the result. For a 7×7 global pooling layer, this allows us to compress the RAM usage to 2% of the original size, without introducing any redundant computations or computation overhead.

Iterative Computation of Dense Layer – We noted that the matrix multiplication in dense layers can be implemented by splitting the input vector into individual elements, multiplying each element with its corresponding weight column, and iteratively summing the results, as shown in Figure 3. Unlike the original approach, which requires the complete input tensor, this method processes only one element of the input tensor per iteration. For a 1024→256 dense layer, this approach compresses memory usage to 20% of the original.

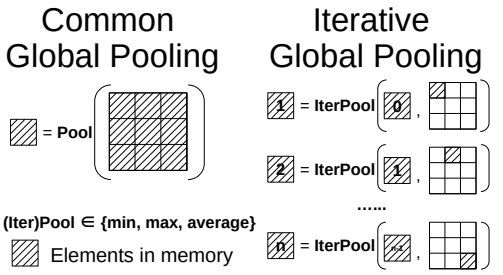


Figure 2: Comparison of common and iterative global pooling.

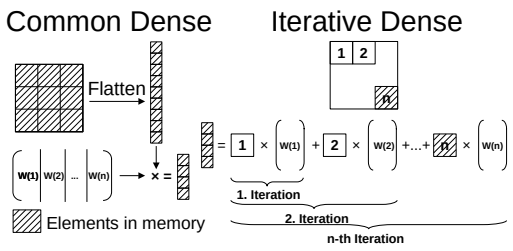


Figure 3: Comparison of common and iterative dense layer. The columns of the weight matrix are denoted as $w(n)$.

8 Experiments on Microcontrollers

In this section we report on experiments running msf-CNN on various MCUs, aiming to validate both the correctness of our optimization strategies and their versatility when applied on diverse hardware.

More concretely, we measured peak RAM usage and compute latency based on the fusion settings in Section 6.3, as reported in the following. As shown in Table 2, we carried out our experiments on the relevant 32-bit microcontroller architectures: Arm Cortex-M, Espressif Xtensa, and RISC-V. For our model zoo, we chose MBV2-w0.35, MN2-vww5 and MN2-320K as they are good representatives of backbones for applications in AIoT [33], as also used in prior works [26, 46]. We compare msf-CNN performance to the closest related work: MCUNetV2 [26] and StreamNet-2D [46], more simply denoted StreamNet in the following.

8.1 Minimal Peak RAM Usage

First, we evaluated solutions to P1 while relaxing Equation (2), i.e. the fusion settings with minimum peak RAM usage, without considering time constraint. Results are shown in Table 3. We observe that, compared to prior art (StreamNet-2D and MCUNetV2), msf-CNN can further reduce the peak RAM usage by 65% to 87%. We could even deploy the MBV2-w0.35 model onto the SiFive board that provides only 16 kB RAM (!). However, achieving this high compression ratio comes at the expense of increased computational latency, which we measured in Table 4. Interestingly, while

Table 2: The different microcontrollers & boards used in our experiments. The RAM and Flash capacity are presented in kB.

Board	MCU	Core	RAM	Flash
Nucleo-f767zi	STM32F767ZI	Cortex-M7 @ 216 MHz	512	2048
Stm32f746g-disco	STM32F746NG	Cortex-M7 @ 216 MHz	320	1024
Nucleo-f412zg	STM32F412ZG	Cortex-M4 @ 100 MHz	256	1024
esp32s3-devkit	ESP32-S3-WROOM-1N8	Xtensa @ 240 MHz	512	8192
esp32c3-devkit	ESP32c3-1-MINI-M4N4	RISC-V @ 160 MHz	384	4096
hifive1b	SiFive FE310-G002	RISC-V @ 320 MHz	16	4096

clock frequency plays a decisive role, MCU architecture can also have a crucial effect, for larger models. For instance, notice latency with Xtensa esp32s3 at 240MHz *versus* RISC-V esp32c3 at 160 MHz, for the MN2-320K model (in Table 4). Nevertheless, we measured that latency increases $2\times$ to $5\times$ compared to vanilla (non-fused) CNN. Hence, such minimal RAM settings are only suitable for latency-tolerant applications on the smallest devices.

Table 3: Minimal peak RAM use, measured in kB. (Vanilla: un-fused model)

(Fusion)	MBV2 -w0.35	MN2 -vww5	MN2 -320K
Vanilla	194.44	96.00	309.76
MCUNetv2	63.00	45.00	215.00
StreamNet	66.00	44.00	208.00
msf-CNN	8.56	15.37	51.16

Table 4: Inference execution time, measured in *ms*, with msf-CNN tuned with minimal peak RAM. (OOM: Out-of-Memory)

(MCU)	MBV2 -w0.35	MN2 -vww5	MN2 -320K
stm32f767	1996.8	1723.0	19329.9
(vs. vanilla)	$2.5\times$	$3.4\times$	$4.4\times$
stm32f746	1379.6	1727.5	16261.9
stm32f412	5270.1	4943.4	56979.0
esp32s3	6748.2	5974.1	76763.6
esp32c3	6792.7	6248.9	73713.8
SiFive	10000.0	OOM	OOM

8.2 Impact of RAM Budget Limit

As shown in Figure 4 and Table 6, the measured peak RAM usage consistently obeys to the given constraints, thereby validating the correctness of the optimizer and corroborating our analytical results. Based on these, we observe that higher RAM budgets result in shorter compute latency for the optimal fusion configurations identified by msf-CNN. This is because the optimizer tends to favor configurations with either no fusion or shallow fusion depths, which correspond to higher peak RAM usage but lower computational costs.

For the MBV2-w0.35 and MN2-vww5 models, our method outperforms MCUNetV2 when the RAM limit is set to 32kB and 64kB. Although our method does not surpass StreamNet-2D across the board, msf-CNN does demonstrate its flexibility, enabling users to select the optimal fusion configuration under varying memory budgets.

8.3 Impact of Computation Cost Limit

When capping computation cost as a constraint, the relation between compute latency and peak RAM usage is consistent (dual) with the previous section, such that higher compute overhead budgets result in longer compute latency and smaller peak RAM usage. We also observe that the ratio F measuring the overhead compared to vanilla CNN (no fusion) is bigger than the F_{max} we set for. This discrepancy comes from the fact that the optimizer computes the amount of MAC operations, whereas the full latency includes not only MAC operations but also I/O delays. In mainstream MCUs, model weights are stored in Flash rather than RAM, which introduces substantial additional latency during read operations, thereby contributing to higher compute latency. Specifically, when recomputation occurs, the weights must be refetched from flash memory, which could disrupt cache hits and lead to increased overall latency. Despite this discrepancy, our method still generates fusion configurations for the MBV2-w0.35 and MN2-vww5 models that outperform MCUNetV2.

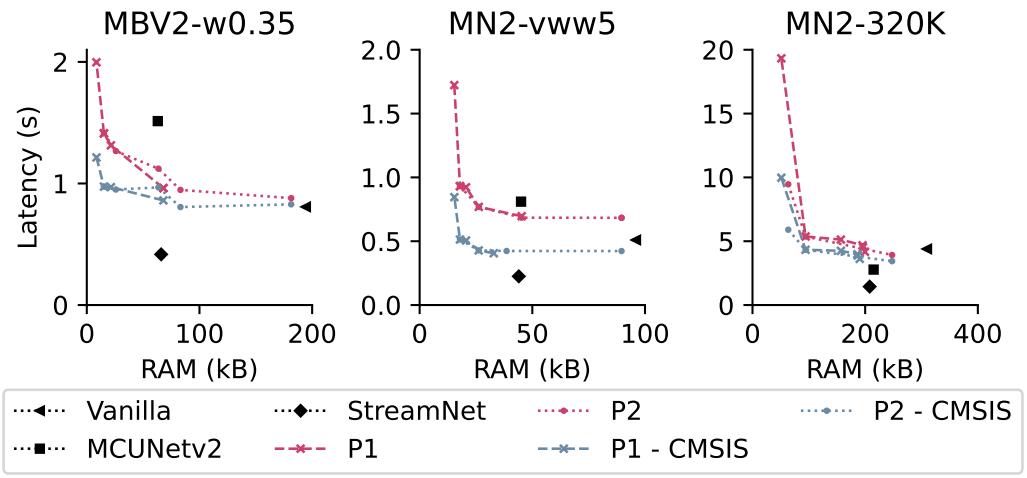


Figure 4: Trade-off between RAM and latency of different optimal fusion settings on Nucleo-f767zi. P1: Minimize RAM s.t. compute cost limit. P2: Minimize compute cost s.t. RAM limit. For more detailed results, please refer to Table 6 in Appendix F.

Particularly for memory-sensitive but time-insensitive applications, we can set the constraint F_{max} to infinity, thereby obtaining novel fusion configurations with minimal RAM usage.

msf-CNN underperforms StreamNet because our implementation does not use hardware-dependent libraries or acceleration instructions, while StreamNet is optimized for ARM platforms via the CMSIS [22] and employs a 2-D cache (ours uses 1-D), which further mitigates recomputation. To enable a fairer comparison, we added an optional CMSIS backend to msf-CNN for ARM devices and conducted additional measurements, as shown in Figure 4. The results show msf-CNN's Pareto curves of CMSIS variants are much closer to StreamNet's. We are also extending msf-CNN with 2-D cache support to provide greater flexibility across hardware platforms.

9 Discussion

Our experiments demonstrate msf-CNN's capability to optimize resource usage with diverse CNN models, under user-specified constraints emphasizing either compute latency or RAM footprint. Furthermore, msf-CNN generates code that is deployable across diverse MCUs' ISAs. Users can thus produce optimal CNN fusion configurations tailored to specific industrial hardware requirements. However, some limitations remain, on which our future work will focus next:

Parameter Space – The current optimization scope is limited to fusion block positioning and depth selection, with the number of output elements per iteration fixed at one. This parameter significantly impacts both memory footprint and compute overhead, which warrants further exploration.

Caching Paradigm – The search space currently incorporates only the H-cache paradigm. Future implementations should integrate alternative caching strategies to enhance optimization flexibility.

Neural Network Architecture – The work currently focuses exclusively on convolutional neural network architectures (CNNs). The analysis of other prevalent structures, particularly attention mechanisms and recurrent neural networks (RNNs), remains an open research direction. Please check Appendix G for the current state of extension.

10 Related Work

Machine Learning Compilers for MCUs – Compilers such as Tensor Virtual Machine (TVM)[7], IREE[39], FlexTensor [48], and Buddy [44] offer automated transpilation and compilation for models produced by major Machine Learning (ML) frameworks, including TensorFlow and PyTorch. Other prior work such as RIOT-ML [17] combine a small general-purpose OS with microTVM (extension of

TVM orienting to MCU), for comprehensive support for ML frameworks and operator implementation on divers MCUs. However, none of the above tools provide CNN fusion optimization mechanisms, in contrast to msf-CNN.

Efficient Neural Network Structure – For models to operate on low-power IoT devices, they must be compact and computationally efficient. Studies have demonstrated the use of lightweight CNNs for speech recognition and age classification [27], water leakage detection [3], fall detection for the elderly [9] and other tasks [18, 50]. Tiny vision transformers have also been employed for classification tasks in various studies [20, 23, 43, 41]. Besides handcrafting a lightweight structure by reducing layer number or kernel size, people [19, 38, 16, 34, 15] also re-designed the basic blocks to replace common convolutions for lower memory footprint and compute latency.

Tiny Neural Architecture Search (NAS) – This technique is employed to automatically search for model structures with optimal accuracy under the constraints of memory, flash footprint and compute latency. TinyNAS [25], μNAS [24] and the Once-for-All Network [6] leverage Neural Architecture Search (NAS) to design CNNs with exceptionally small memory requirements for MCUs. The resulting networks require only a few hundred kilobytes of RAM for execution. However, contrary to msf-CNN, these methods necessitate retraining or fine-tuning of pre-existing networks.

Memory Optimization for CNN layers – Memory optimization strategies can be broadly categorized into scheduling-based and fusion-based methods. Scheduling-based methods, such as those implemented in frameworks like Ansor [47], vMCU [49], MoDEL [37] and TinyEngine [26], focus on the efficient reuse of memory pools to minimize peak memory usage by leveraging the different lifetimes of inter- and intra-layer tensors. Although both methods achieve a peak memory reduction exceeding 50%, they still generate a complete output tensor for each layer. This requirement remains problematic for low-power MCUs with limited RAM. Prior work on fusion was covered in Section 2. Contrary to msf-CNN, these methods do not fully exploit the potential of multiple fusion blocks.

11 Conclusion

Convolutional neural networks (CNNs) must not only execute in the cloud or on edge computing gateways, but also on the smaller, more energy-efficient microcontroller-based devices which take part in our cyber-physical systems. Microcontrollers pose a great challenge for CNNs regarding the joint optimization of RAM memory consumption and inference latency. In this context, we presented msf-CNN, a technique and heuristics able to identify pools of practical patch-based fusion optimizations for CNN inference, which jointly satisfy memory and latency constraints. Compared to previous work on CNN fusion for microcontrollers, msf-CNN identifies a wider set of applicable solutions, on much more diverse hardware. Our experimental evaluation using the open source implementation we provide for common microcontrollers (ARM Cortex-M, RISC-V, and ESP32) show that msf-CNN can achieve inference with less than 50% the peak RAM usage compared to state-of-the-art. As such, msf-CNN provides a new level of flexibility for embedded system designers, which can now better tune the trade-off between peak RAM and model inference latency on various hardware.

Acknowledgment

The authors thank Adrien Tousnakhoff for useful discussions. Work contributing to these results was partially funded by the ANR France 2030 Programme (ANR-22-PTCC-0001 and ANR-22-PEFT-0007).

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

[2] Alwani, M., Chen, H., Ferdman, M., and Milder, P. Fused-layer CNN accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, Taipei,

Taiwan, October 2016. IEEE. ISBN 978-1-5090-3508-3. doi: 10.1109/MICRO.2016.7783725. URL http://ieeexplore.ieee.org/document/7783725/.

[3] Atanane, O., Mourhir, A., Benamar, N., and Zennaro, M. Smart Buildings: Water Leakage Detection Using TinyML. *Sensors*, 23(22):9210, November 2023. ISSN 1424-8220. doi: 10.3390/s23229210. URL https://www.mdpi.com/1424-8220/23/22/9210.

[4] Baccelli, E., Gündoğan, C., Hahm, O., Kietzmann, P., Lenders, M. S., Petersen, H., Schleiser, K., Schmidt, T. C., and Wählisch, M. RIOT: An Open Source Operating System for Low-end Embedded Devices in the IoT. *IEEE Internet of Things Journal*, 5(6):4428–4440, 2018.

[5] Bormann, C. et al. Terminology for Constrained-Node Networks. RFC 7228, May 2014.

[6] Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[7] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., Shen, H., Wang, L., Hu, Y., Ceze, L., et al. Tvm: An automated end-to-end optimizing compiler for deep learning. *arXiv preprint arXiv:1802.04799*, 2018.

[8] Dijkstra, E. W. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: his life, work, and legacy*, pp. 287–290. 2022.

[9] Fang, K., Xu, Z., Li, Y., and Pan, J. A Fall Detection using Sound Technology Based on TinyML. In *2021 11th International Conference on Information Technology in Medicine and Education (ITME)*, pp. 222–225, Wuyishan, Fujian, China, November 2021. IEEE. ISBN 978-1-66540-679-6. doi: 10.1109/ITME53901.2021.00053. URL https://ieeexplore.ieee.org/document/9750658/.

[10] Fredman, M. L. and Tarjan, R. E. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[11] Ghosh, A., Chakraborty, D., and Law, A. Artificial intelligence in internet of things. *CAAI Transactions on Intelligence Technology*, 3(4):208–218, 2018.

[12] Guo, M.-H., Lu, C.-Z., Hou, Q., Liu, Z., Cheng, M.-M., and Hu, S.-M. Segnext: Rethinking convolutional attention design for semantic segmentation. *Advances in neural information processing systems*, 35:1140–1156, 2022.

[13] Guo, M.-H., Lu, C.-Z., Liu, Z.-N., Cheng, M.-M., and Hu, S.-M. Visual attention network. *Computational visual media*, 9(4):733–752, 2023.

[14] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[15] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for MobileNetV3. pp. 1314–1324, . URL https://openaccess.thecvf.com/content_ICCV_2019/html/Howard_Searching_for_MobileNetV3_ICCV_2019_paper.html.

[16] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications, . URL http://arxiv.org/abs/1704.04861.

[17] Huang, Z., Zandberg, K., Schleiser, K., and Baccelli, E. RIOT-ML: toolkit for over-the-air secure updates and performance evaluation of TinyML models. *Annals of Telecommunications*, pp. 1–15, 2024.

[18] Hussain, M. S. and Haque, M. A. SwishNet: A Fast Convolutional Neural Network for Speech, Music and Noise Classification and Segmentation. 2018. doi: 10.48550/ARXIV.1812.00149. URL https://arxiv.org/abs/1812.00149.

[19] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size.

[20] Jinyang Yu, Zikai Song, Jiahao Ji, Lixian Zhu, Kele Xu, Qian, K., Dou, Y., and Hu, B. Tiny Audio Spectrogram Transformer: Mobilevit for Low-Complexity Acoustic Scene Classification with Decoupled Knowledge Distillation. 2023. doi: 10.13140/RG.2.2.24001.12646. URL https://rgdoi.net/10.13140/RG.2.2.24001.12646.

[21] Koonce, B. Resnet 34. *Convolutional neural networks with swift for tensorflow: image recognition and dataset categorization*, pp. 51–61, 2021.

[22] Lai, L., Suda, N., and Chandra, V. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*, 2018.

[23] Liang, Y., Wang, Z., Xu, X., Tang, Y., Zhou, J., and Lu, J. MCUFormer: Deploying Vision Transformers on Microcontrollers with Limited Memory. 2023. doi: 10.48550/ARXIV.2310.16898. URL https://arxiv.org/abs/2310.16898.

[24] Liberis, E., Dudziak, Ł., and Lane, N. D. μnas: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pp. 70–79, 2021.

[25] Lin, J., Chen, W.-M., Lin, Y., Gan, C., Han, S., et al. Mcunet: Tiny deep learning on iot devices. *Advances in neural information processing systems*, 33:11711–11722, 2020.

[26] Lin, J., Chen, W.-M., Cai, H., Gan, C., and Han, S. Memory-efficient Patch-based Inference for Tiny Deep Learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 2346–2358. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/1371bccec2447b5aa6d96d2a540fb401-Abstract.html.

[27] Maayah, M., Abunada, A., Al-Janahi, K., Ahmed, M. E., and Qadir, J. LimitAccess: on-device TinyML based robust speech recognition and age classification. *Discover Artificial Intelligence*, 3(1):8, February 2023. ISSN 2731-0809. doi: 10.1007/s44163-023-00051-x. URL https://link.springer.com/10.1007/s44163-023-00051-x.

[28] Mei, L., Goetschalckx, K., Symons, A., and Verhelst, M. 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)*, pp. 570–583. IEEE, 2023.

[29] Niu, W., Guan, J., Wang, Y., Agrawal, G., and Ren, B. Dnnfusion: accelerating deep neural networks execution with advanced operator fusion. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 883–898, 2021.

[30] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[31] Pinckaers, H., van Ginneken, B., and Litjens, G. Streaming Convolutional Neural Networks for End-to-End Learning With Multi-Megapixel Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3):1581–1590, March 2022. ISSN 1939-3539. doi: 10.1109/TPAMI.2020.3019563. URL https://ieeexplore.ieee.org/document/9178453. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[32] Robert, S. Algorithms in c, part 5: Graph algorithms, 2002.

[33] Saha, S. S., Sandha, S. S., and Srivastava, M. Machine learning for microcontroller-class hardware: A review. *IEEE Sensors Journal*, 22(22):21362–21390, 2022.

[34] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobileNetV2: Inverted residuals and linear bottlenecks. pp. 4510–4520. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html.

[35] Sedgewick, R. *Algorithms in c, part 5: graph algorithms, third edition*. Addison-Wesley Professional, third edition, 2001. ISBN 9780768685329.

[36] Shen, Z., Zhang, M., Zhao, H., Yi, S., and Li, H. Efficient attention: Attention with linear complexities. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pp. 3531–3539, 2021.

[37] Steiner, B., Elhoushi, M., Kahn, J., and Hegarty, J. Model: memory optimizations for deep learning. In *International Conference on Machine Learning*, pp. 32618–32632. PMLR, 2023.

[38] Tan, M. and Le, Q. V. EfficientNet: Rethinking model scaling for convolutional neural networks. URL http://arxiv.org/abs/1905.11946.

[39] The IREE Authors. IREE, September 2019. URL https://github.com/iree-org/iree.

[40] Wang, G., Lin, Y., and Yi, W. Kernel fusion: An effective method for better power efficiency on multithreaded gpu. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pp. 344–350. IEEE, 2010.

[41] Wyatt, S., Elliott, D., Aravamudan, A., Otero, C. E., Otero, L. D., Anagnostopoulos, G. C., Smith, A. O., Peter, A. M., Jones, W., Leung, S., and Lam, E. Environmental Sound Classification with Tiny Transformers in Noisy Edge Environments. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, pp. 309–314, June 2021. doi: 10.1109/WF-IoT51360.2021.9596007. URL https://ieeexplore.ieee.org/abstract/document/9596007.

[42] Xingjian, S. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28:1, 2015.

[43] Yao, Z. and Liu, X. A CNN-Transformer Deep Learning Model for Real-time Sleep Stage Classification in an Energy-Constrained Wireless Device *. In *2023 11th International IEEE/EMBS Conference on Neural Engineering (NER)*, pp. 1–4, Baltimore, MD, USA, April 2023. IEEE. ISBN 978-1-66546-292-1. doi: 10.1109/NER52421.2023.10123825. URL https://ieeexplore.ieee.org/document/10123825/.

[44] Zhang, H., Xing, M., Wu, Y., and Zhao, C. Compiler technologies in deep learning co-design: A survey. *Intelligent Computing*, 2:0040, 2023.

[45] Zhao, J., Gao, X., Xia, R., Zhang, Z., Chen, D., Chen, L., Zhang, R., Geng, Z., Cheng, B., and Jin, X. Apollo: Automatic partition-based operator fusion through layer by layer optimization. *Proceedings of Machine Learning and Systems*, 4:1–19, 2022.

[46] Zheng, H.-S., Liu, Y.-Y., Hsu, C.-F., and Yeh, T. T. Streamnet: memory-efficient streaming tiny deep learning inference on the microcontroller. *Advances in Neural Information Processing Systems*, 36, 2024.

[47] Zheng, L., Jia, C., Sun, M., Wu, Z., Yu, C. H., Haj-Ali, A., Wang, Y., Yang, J., Zhuo, D., Sen, K., et al. Ansor: Generating {High-Performance} tensor programs for deep learning. In *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, pp. 863–879, 2020.

[48] Zheng, S., Liang, Y., Wang, S., Chen, R., and Sheng, K. Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 859–873, 2020.

[49] Zheng, S., Chen, R., Li, M., Ye, Z., Ceze, L., and Liang, Y. vMCU: Coordinated Memory Management and Kernel Optimization for DNN Inference on MCUs. *Proceedings of Machine Learning and Systems*, 6:452–464, May 2024.

[50] Zhu-Zhou, F., Tejera-Berengué, D., Gil-Pita, R., Utrilla-Manso, M., and Rosa-Zurera, M. Computationally constrained audio-based violence detection through transfer learning and data augmentation techniques. *Applied Acoustics*, 213:109638, October 2023. ISSN 0003-682X. doi: 10.1016/j.apacoust.2023.109638. URL https://www.sciencedirect.com/science/article/pii/S0003682X2300436X.

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 main claims in the abstract and introduction reflect the contribution and scope of the paper.

   Guidelines:

   - The answer NA 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 NA 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 are presented in Section 9.

   Guidelines:

   - The answer NA 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: We transformed the optimization problems presented in Section 4 into shortest path problems (or the variants) and solved them by classic graph algorithm like Dijkstra's, whose correctness is proven by predecessors. The time complexity is given and proven in Appendix D.

Guidelines:

- The answer NA 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 technical details are provided in the experimental parts and in the supplemental appendix.

Guidelines:

- The answer NA 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: We open-sourced the code in a anonymous repository.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.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://nips.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, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We released all technical details throughout the paper and the code.

   Guidelines:

   - The answer NA 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: All results are deterministic. No need to report error bars.

   Guidelines:

   - The answer NA 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: The information is provided in Section 8 and Appendix D.

   Guidelines:

   - The answer NA 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: The research is aligned with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA 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: [NA]

    Justification: This work is focused on technical improvements to the core algorithmic framework and does not directly address an application area with immediate societal implications.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not release any models or data.

    Guidelines:

    - The answer NA 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 used assets are properly cited in the paper.

    Guidelines:

    - The answer NA 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: We released our code in a anonymous repository.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA 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 rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The LLMs were only used for wordsmiths.

Guidelines:

- The answer NA 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 (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Relation between msf-CNN and kernel fusion

We note that readers might potentially confuse patch-based multi-stage fusion (msf-CNN, our approach) on the one hand, and on the other hand traditional kernel fusion techniques. These two approaches are orthogonal and can be applied concurrently for maximum benefit.

While kernel fusion optimizes computation overhead, msf-CNN instead targets memory efficiency, the latter being the critical first hurdle on small edge devices. As such, we disambiguate this further:

Kernel fusion [40, 29, 45] focuses primarily on reducing redundant data movements between GPU and RAM by combining multiple primitive operators (e.g. Batch Normalization, ReLU, Softmax, etc.) with a primary, memory-bound operator (e.g. conv, pooling) into a single kernel. While kernel fusion improves compute latency and data throughput, it does not address the fundamental memory usage issue that arises when processing multiple primary operators sequentially.

Patch-based multi-stage fusion (msf-CNN, our approach) extends the idea of patch-based fusion [2, 28]. More specifically, msf-CNN:

- Fuses multiple layers (i.e. primary operators like convolution and pooling) into a single computational stage. Implements patch-based partial computation, which drastically reduces peak memory usage by processing input data in smaller patches while maintaining accuracy.
- Introduces a compute-memory trade-off mechanism that allows users to prioritize either memory consumption or computational efficiency based on their deployment constraints.

This makes msf-CNN fundamentally different from traditional kernel fusion techniques.

To the best of our knowledge, the closest related works to msf-CNN are StreamNet and MCUNetv2, which have been the state-of-the-art for patch-based fusion on microcontrollers so far. Compared to this state of the art, based on our measurements, msf-CNN achieves up to 50% reduction of peak RAM usage in model inference, for the same inference accuracy, which thus enables more models to fit smaller devices.

B Analysis of the H-cache buffer size

For a fusion block containing n layers, the cache buffer size of the i-th layer under H-cache scheme is given by

$$\text{Buf}_i = t_i \times k_i \times c_i^{in} \tag{11}$$

where t_i, k_i and c_i^{in} are the tile size, kernel size and input channels number, respectively. Obviously, the first layer of the fusion block does not need any input cache, thus $\text{Buf}_1 = 0$. The total cache size of the fusion block is $\text{Buf} = \sum_i \text{Buf}_i$.

C Analysis of the amount of MAC operations

Analyzing the number of MAC operations in the fusion block is quite complex. The input tensor for each layer is sliced into overlapped tiles, and the kernel performs convolution on the data within each tile. Here, the number of overlapped tiles N^{tile} of each layer is

$$N^{tile} = \lfloor \frac{h^{in} + 2p - t}{s^{tile}} + 1 \rfloor \lfloor \frac{w^{in} + 2p - k}{s^{layer}} + 1 \rfloor, \tag{12}$$

where h^{in}, w^{in} are the height and width of input tensor, s^{tile}, s^{layer} are the stride of tile and layer, p represents the input padding. Recall that t, k are the tile size and kernel size respectively.

And the output size of each tile is determined as:

$$O^{tile} = \lfloor \frac{t - k}{s^{layer}} + 1 \rfloor c^{out}. \tag{13}$$

whereby c^{out} is the number of output channels. We can therefore derive the number C^{layer} of MAC operations of a fused convolutional layer as:

$$C^{layer} = N^{tile} \times O^{tile} \times k^2 \times c^{out}. \tag{14}$$

Finally, we can derive C^{fb} the total MAC operations of the entire fusion block as:

$$C^{fb} = \sum C^{layer}. \tag{15}$$

D Complexity analysis of the search algorithm

We provide a quick preliminary analysis of the *worst-case scenario*. We also highlight that these shortest path computations do not take place on the microcontroller at runtime, but offline on a PC (which expands the realm of what can be assessed as bearable computation).

First, we consider the *lower-bound* of the search algorithm. As shown in Section 6, both Problems P1 and P2 without constraints can be transformed into a multiple single-source-single-target shortest path problem, which can by solved by Dijkstra's algorithm with Fibonacci heap [10] with complexity:

$$O(E + Vlog(V)),$$

where E and V denote the edges (possible fusion blocks) and vertices (layers) of the DAG. In the worst case $E = \sum_{n=1}^{V}(n-1)$, leading the *overall lower-bound* to $O(V^2)$.

Concerning Problem P1 with constraints: if we don't prune the search space (iteratively), we need to brute force all possible fusion settings of the DAG to form a subspace that fulfills the latency constraints. This leads to enumerating all simple paths from input layer to the output layer. In the worst-case, starting from the input layer, we obtain 2^{V-2} fusion combinations, which becomes unbearable for a deep neural network. We can prove this complexity by simple induction:

In worst-case scenario, we assume a complete DAG, where each node is connected to all its predecessors.

Base Case: For $V = 2$, there is only one complete compute path:

$$2^{V-2} = 2^0 = 1.$$

Inductive Step: Assume a complete DAG with k nodes has 2^{k-2} complete compute paths. To construct a complete DAG with $k+1$ nodes:

1. Add a new node with the same incoming edges as the last node. This contributes 2^{k-2} paths.
2. Connect the new node to the duplicated node, adding another 2^{k-2} paths.

Total paths:

$$2^{k-2} + 2^{k-2} = 2^{k-1}.$$

Therefore, by induction, a complete DAG with V nodes has 2^{V-2} complete compute paths. ■

Hence, we apply a pruning strategy (see Equ. 9-11) to reduce the complexity from $O(2^{V-2})$ to $O(V^3)$. The idea: we erase the edges with maximal RAM usage per iteration. In the worst-case, only one edge is erased in each iteration, with a complete DAG with $\frac{V(V-1)}{2}$ edges. Thus, the worst-case complexity of our search algorithm for constrained Problem P1 is $O(V^3)$.

E Accuracy Evaluation

msf-CNN is a computation scheduling and memory optimization technique that does not alter the model's architecture, parameters, or the mathematical operations performed. Like MCUNet and StreamNet, msf-CNN only changes when and how intermediate results are computed and stored to minimize peak memory. Therefore, the final output, and consequently the model's accuracy, remains identical to the original, unfused model. Hence, standard ML performance benchmarks focusing on accuracy are irrelevant here.

Nevertheless, to make sure, we conducted extra experiments on imagenet and vww dataset, as MCUNet did. We reused the model weights from the pre-trained MCUNet models, and selected

Table 5: Model Top-1 Accuracy

Method	MBV2-w0.35	MN2-vww5	MN2-320K
Vanilla	48.94 %	88.89 %	61.76 %
msf-CNN	48.94 %	88.89 %	61.76 %

$P_{max} = 64kB$ as fusion constraints. The results show in Table 5 that no Top-1 accuracy drop was found between vanilla models and the msf-CNN variants.

F Experiment Details

Here, we provide additional details of the experiments, including results not presented in the main text.

F.1 Supplemental Results

Table 6 provides the underlying data corresponding to Figure 4, including the specific constraints applied to P1 and P2, respectively.

Table 6: Optimal fusion settings on Nucleo-f767zi. RAM (kB), Latency (ms). SAA: Same as above. Gray: msf-CNN beats MCUNetv2.

		MBV2-w0.35		MN2-vww5		MN2-320K	
		RAM	Latency	RAM	Latency	RAM	Latency
Vanilla		194.44	807.60	96.00	509.70	309.76	4394.30
MCUNetv2		63.00	1513.00	45.00	810.00	215.00	2777.00
StreamNet		66.00	417.00	44.00	225.00	208.00	1444.00
P1: Min. RAM s.t. Compute Cost Limit							
F_{max}	1.1	68.00	961.90	45.28	696.00	199.60	4171.00
	1.2	(SAA)		26.24	769.20	196.07	4525.10
	1.3	21.39	1313.80	20.57	922.70	195.33	4680.70
	1.4	15.20	1412.30	17.90	931.30	156.86	5128.90
	1.5	(SAA)		(SAA)		94.22	5370.30
	Inf	8.56	1996.80	15.37	1723.00	51.16	19 329.90
P2: Min. Compute Cost s.t. RAM Limit							
P_{max}	16 kB	15.20	1412.30	17.90	931.30	(No Solution)	
	32 kB	25.80	1266.30	26.24	769.20		
	64 kB	63.60	1121.70	45.28	684.60	63.46	9458.60
	128 kB	83.13	947.00	89.60	683.40	94.22	5370.30
	256 kB	181.44	879.20	(SAA)		247.81	3923.20

F.2 Impact on Power Consumption

We used the Nordic Semiconductor Power Profiler Kit II (PPK2) on nrf52840 boards to provide preliminary measurements of the power draw. We observed that across all optimization configurations, inference current and deep sleep current remained consistently around 5.4 mA and 1.9 mA, respectively, which hints at a straightforward link between latency and energy consumption. We note, however, that the precision of the inference current we measured may be limited due to our measurement setup and should be subject to further investigations.

G Network Architecture Extension

We are currently extending msf-CNN to support RNN- and Transformer-based architecture.

G.1 msf-CNN on RNN/LSTM/GRU

For 1-D sequence input, it is trivial to fuse the cascade RNN cells into one block, to avoid outputting the complete sequence between RNNs during inference. This eliminates RAM usage for storing intermediate output, especially when facing a long input sequence. The compute cost remains the same as the original models, since no recompute was introduced compared to 2-D inputs.

For 2-D input [42], all matrix-vector multiplications are replaced by 2-D convolutions and the Hadamard product is applied on two matrices inside the RNN cells. In this case, we can analyze the compute graph to check which convolution operators (inter- or intra RNN cells) can be fused into blocks, and calculate their RAM usage, compute cost as we did in the original msf-CNN.

G.2 msf-CNN on Transformer

We only considered decoder-only architecture so far.

For 1-D sequence input, there is not much optimization space, since the transformer/attention blocks are full of dense layers or heavy matrix-matrix multiplications, where patch-based fusion cannot be applied to reduce RAM usage. On the other hand, we could apply msf-CNN on the covolutional layers before or after transformer blocks of hybrid model (such as in ViT [13]), and integrate Efficient Attention [36] to further improve space and compute complexity of attention layer.

For 2-D image input, msf-CNN can be extended to the convoluional variants of common attention layers (such as VAN [13] and SegNeXt [12]) which contain cascade convolutional layers inter- or intra the attention layers. In this case, msf-CNN can be applied on deciding the fusion strategy.