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

on MLaftermath Workshop

Zhaolan Huang & Emmanuel Baccelli

Freie Universität Berlin & Inria Berlin & Einstein Center Digital Future

9. Mar. 2026

Agenda

- (Tiny) Machine Learning
- Challenges in TinyML
- Problem Statement
- Encoding Costs in DAG
- Implementation Details
- Experiments on MCUs
- Conclusion
- On-going Work

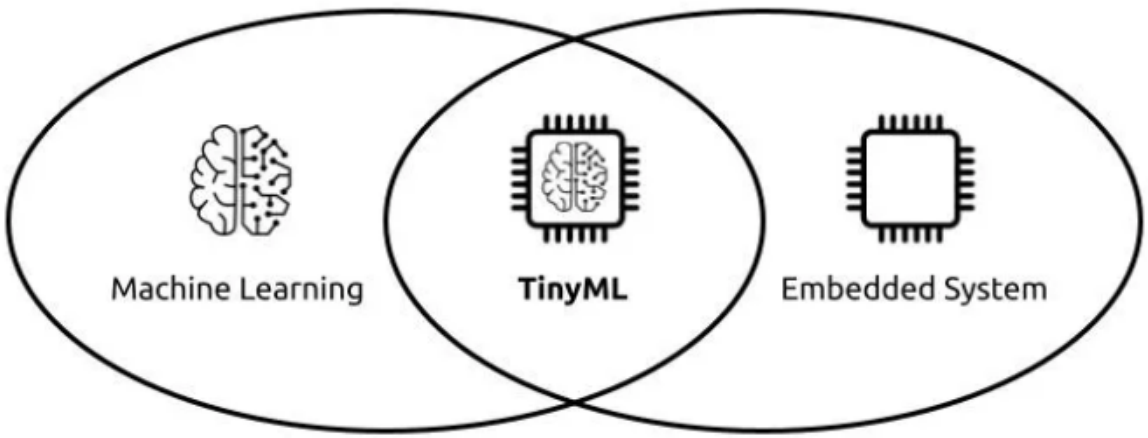
(Tiny) Machine Learning

Machine Learning (ML)

- Complex, compute-intensive algorithms.
- Data-driven decision making.
- Most popular model: (Deep) Neural Network.

Tiny Machine Learning (TinyML)

- Complex, compute-intensive algorithms.
- Data-driven decision making.
- Most popular model: (Deep) Neural Network.
- Deploy on resource-constrained devices (Memory in kB, CPU in MHz).



TinyML: Machine Learning + Embedded System

Agenda

- (Tiny) Machine Learning
- Challenges in TinyML
- Problem Statement
- Encoding Costs in DAG
- Implementation Details
- Experiments on MCUs
- Conclusion
- On-going Work

Challenges in TinyML

So, that elephant will be stuffed into tiny devices...

Yes,

ChatGPT Claude Gemini

Qwen deepseek Z.ai

Mistral AI EURO LLM ALEPH ALPHA

Billions of Parameters; Training / Inference on (Ten)Thousands of GPUs/NPUs

But

264KB Memory
125 MHz MCU

Challenges in TinyML

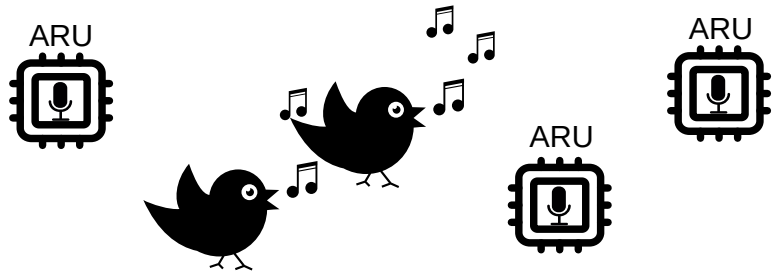
So, that elephant will be stuffed into tiny devices...

- Resource Constraints: RAM, Processor(s), storage.
- Real-time Processing: Inference latency in critical applications.
- Power usage of battery-powered devices...etc.
- While: Keeping good task performance.

Challenges in TinyML

A concrete use-case:

- TinyChirp [1] microcontroller-based **Autonomous Recording Units** (ARU) monitoring birds species in rural UK.



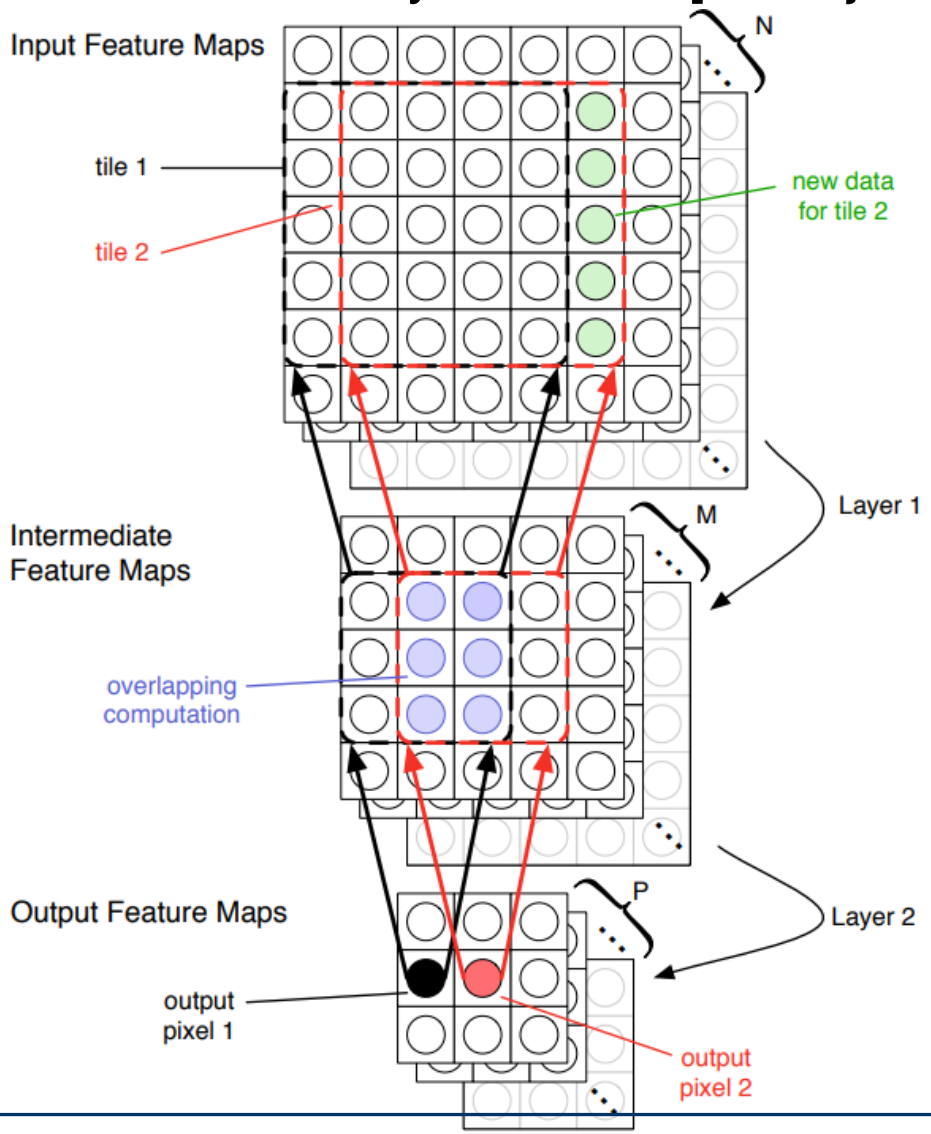
Challenges

- Detecting specific bird song (corn bunting) with high accuracy requires advanced machine learning models, while
- Keeping eyes on resource usage and energy footprint on low-power IoT devices (CPU in MHz, RAM/Flash in KiB/MiB), and
- Satisfying real-time requirements (to not lose audio samples)

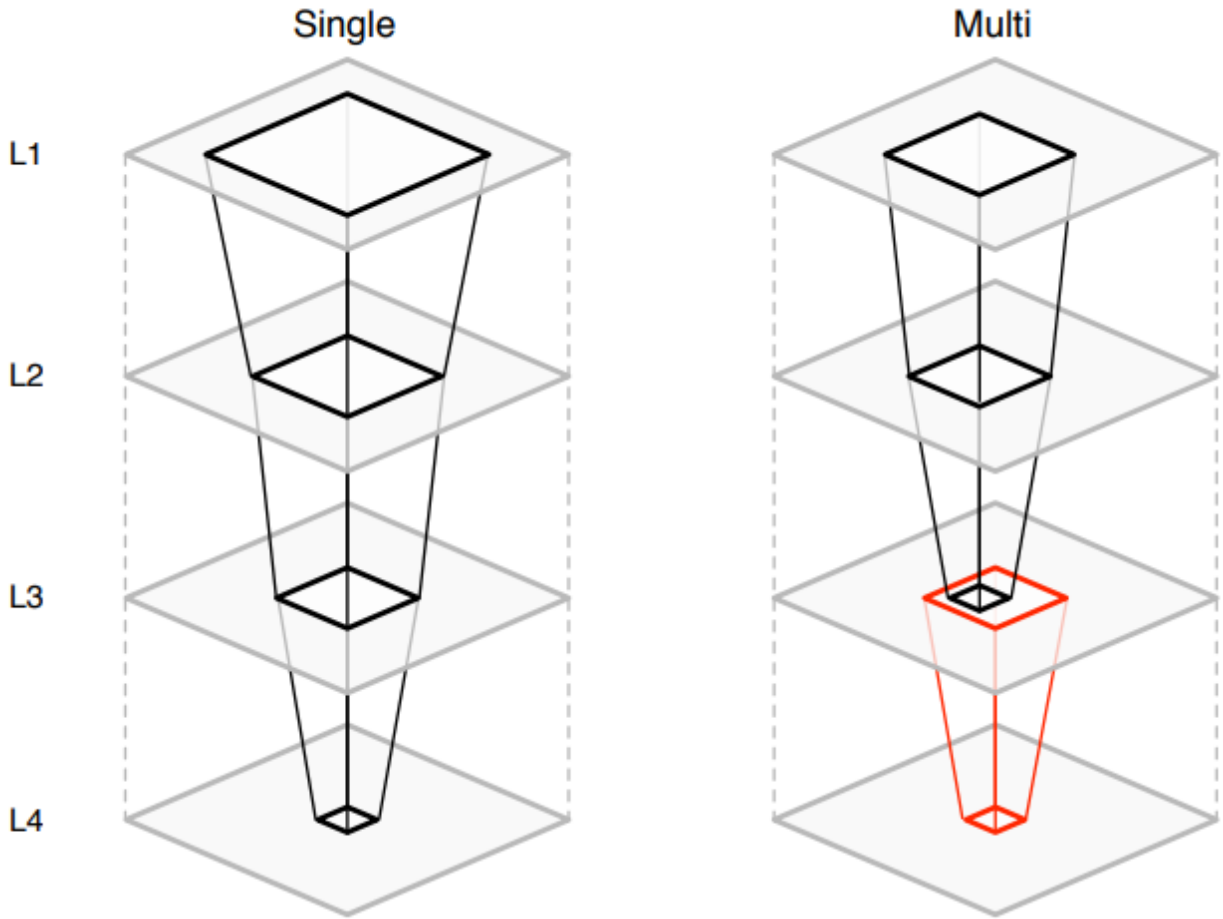
[1] Z. Huang et al. TinyChirp: Bird Song Recognition Using TinyML Models on Low-power Wireless Acoustic Sensors. IEEE IS2, 2024

Patch-based Fusion

Possible Solution: Patch-based Layer Fusion [Manoj Alwani et al, 2016]



Squeezing more RAM: Multi-stage Fusion

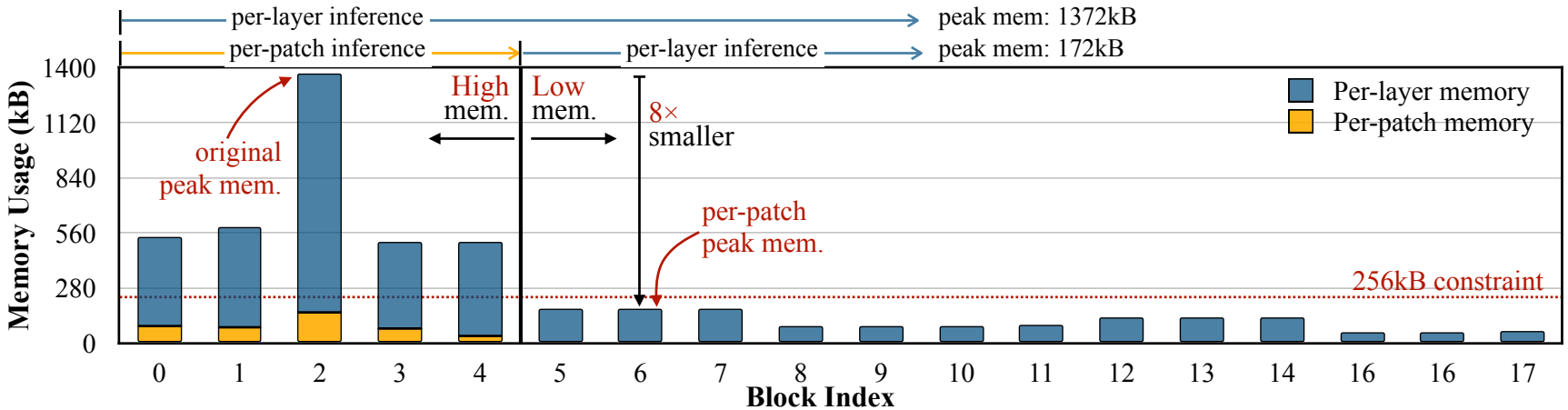


[Manoj Alwani et al, 2016]

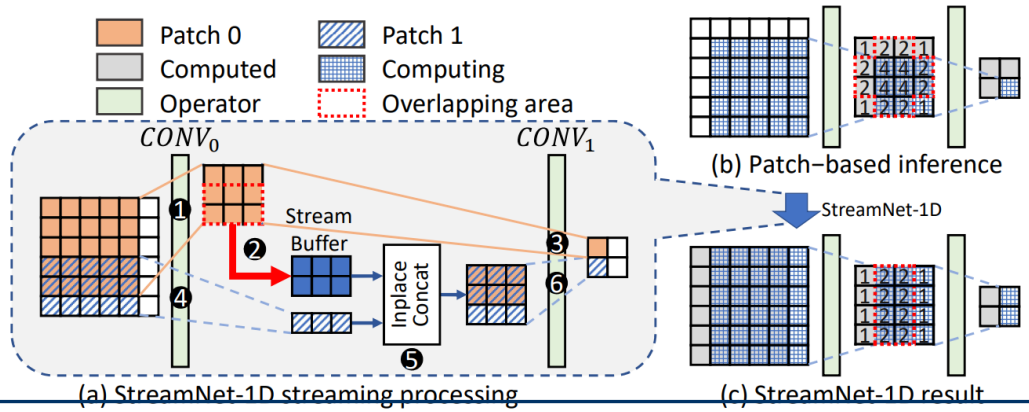
Patch-based Fusion

SOTAs before msf-CNN:

- MCUNetV2: Fuse the heading layers that dominate the RAM usage.



- StreamNet: Relieve the re-compute issue of MCUNetV2 by streaming buffer.



Patch-based Fusion

Limitations of the SOTAs:

- Only fuse the heading layers, DID NOT explore the potential of multiple fusion blocks in CNNs;
- Produce only single solution, rather to provide trade-off space between peak RAM and inference latency for system designers;
- Implementations of fusion on MCUs are very hardware-specific (e.g. bound to the ARM Cortex-M7 instruction set).

Agenda

- (Tiny) Machine Learning
- Challenges in TinyML
- Problem Statement
- Encoding Costs in DAG
- Implementation Details
- Experiments on MCUs
- Conclusion
- On-going Work

Problem Statement

However, the more you fuse, the more you (re-)compute.
Thus, msf-CNN aims to answer:

1. Where to fuse and how to determine the fusion position/depth?
2. Under specific resource constraints, how to find the optimal fusion settings S in all possible fusion options χ (search space)?

Problem Statement

However, the more you fuse, the more you (re-)compute.
Thus, msf-CNN aims to answer:

1. Where to fuse and how to determine the fusion position/depth?
2. Under specific resource constraints, how to find the optimal fusion settings S in all possible fusion options χ (search space)?

Formalization: Dual Optimization Problems

P1: Min. peak RAM usage P subject to maximal compute latency F_{max}

$$\min_{S} P(\chi, S) \text{ s.t. } F(\chi, S) < F_{max} \tag{1}$$

P2: Min. compute latency F subject to maximal peak RAM usage P_{max}

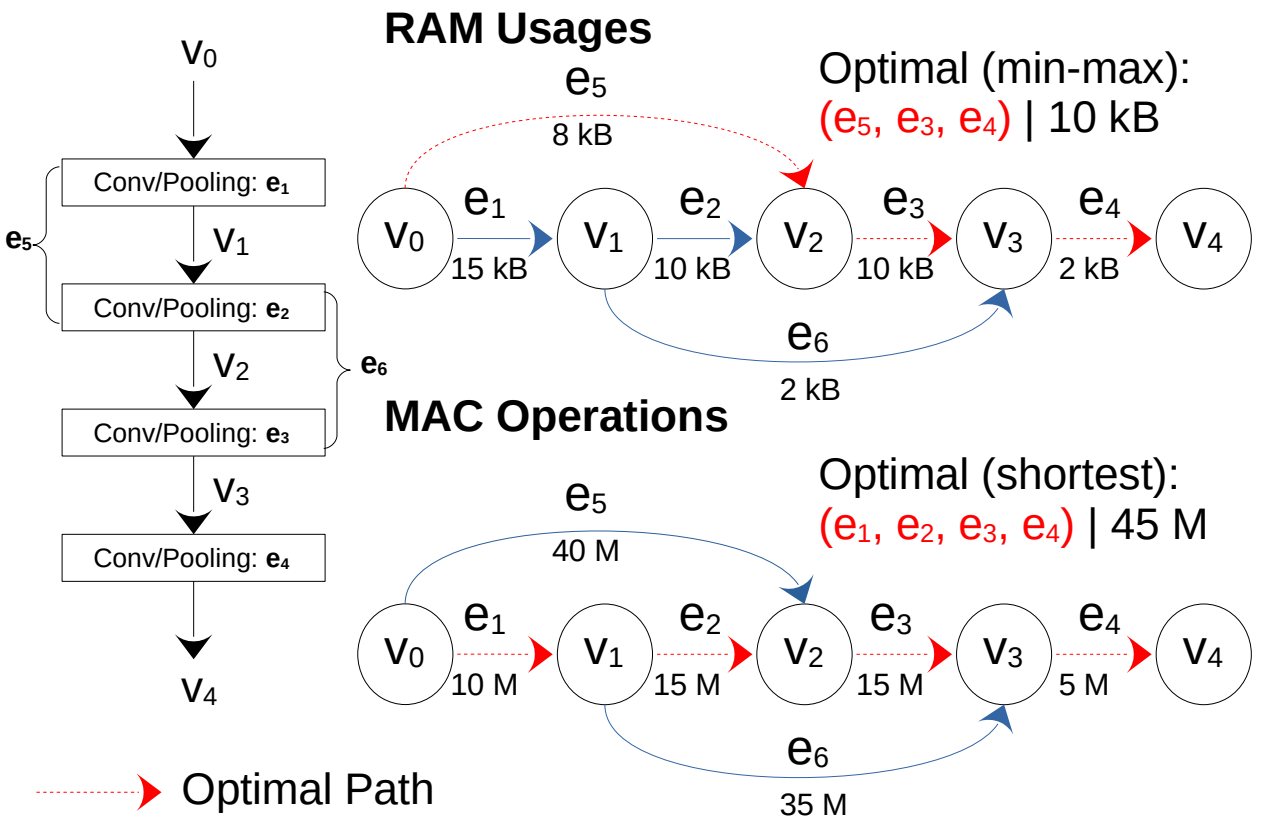
$$\min_{S} F(\chi, S) \text{ s.t. } P(\chi, S) < P_{max} \tag{2}$$

Agenda

- (Tiny) Machine Learning
- Challenges in TinyML
- Problem Statement
- Encoding Costs in DAG
- Implementation Details
- Experiments on MCUs
- Conclusion
- On-going Work

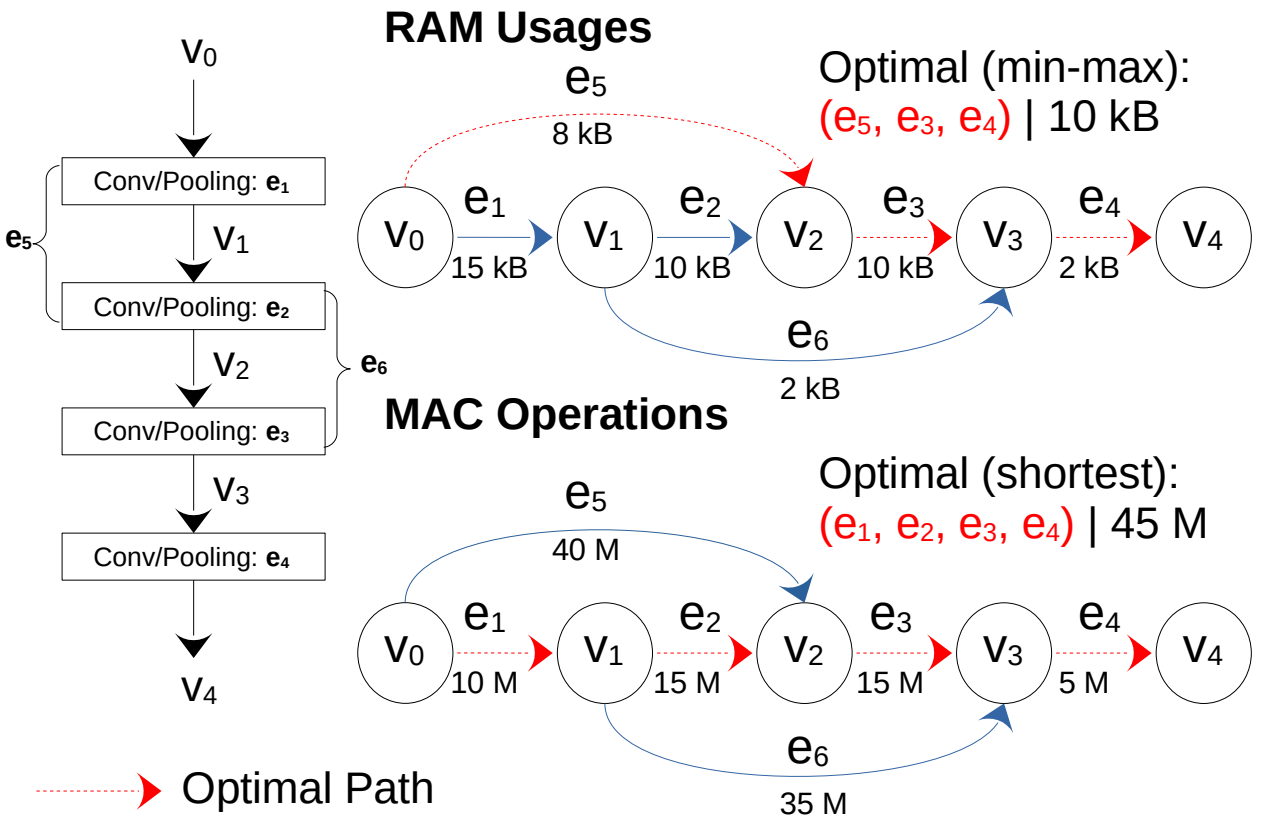
Encoding Costs in DAG

- Neural Networks are modeled as directed acyclic graphs (DAGs)
- P1: Find min-max path under constraints. P2: Find shortest path under constraints. Both problems can be solved in $\mathcal{O}(E \log(V))$ if no constraint applies.



Encoding Costs in DAG

- Problem: Search space bumps to $\mathcal{O}(2^{V-2})$, considering the optimization constraints.



Searching for Optimal Settings

P1 (Min. RAM st. Compute Cost): Iteratively pruning the search space

$$S_i = \arg\min_S F(G_i, S), \tag{3}$$

$$G_i := \text{subgraph of } G_{i-1}, \text{ obtained by removing all edges in } G_{i-1} \text{ with the maximal RAM usage}, \tag{4}$$

$$G_0 = G \tag{5}$$

P2 (Min. Compute Cost st. RAM): Eliminating all edges with encoded RAM usages exceeding the limit.

Reducing Complexity: $\mathcal{O}(2^{V-2})$ to $\mathcal{O}(V^3)$

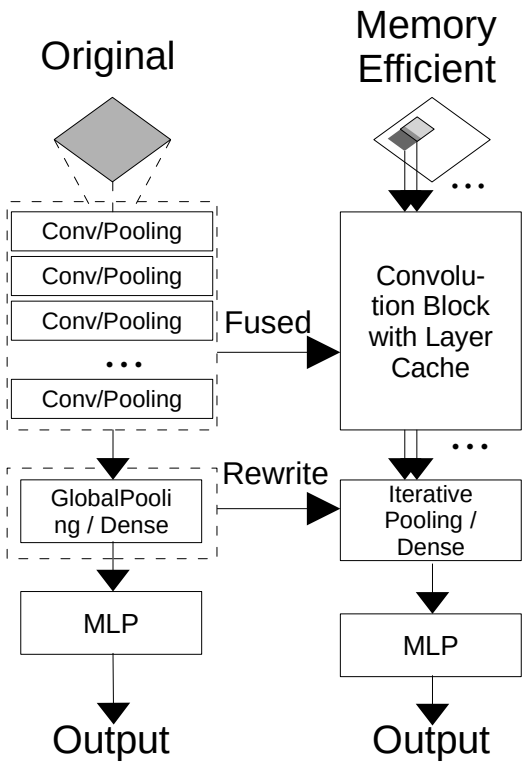
The search algorithms are run offline on PC, not on MCUs.

Agenda

- (Tiny) Machine Learning
- Challenges in TinyML
- Problem Statement
- Encoding Costs in DAG
- Implementation Details
- Experiments on MCUs
- Conclusion
- On-going Work

Implementation Details

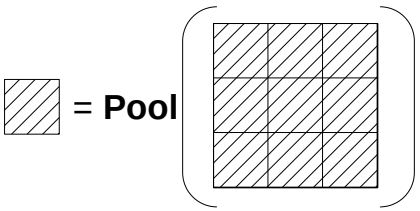
- Implementation leverages RIOT for hardware abstraction and OS services on various MCU hardware (RISC-V, ESP32, Cortex-M)
- Use of generic tinyML pipeline provided by RIOT-ML toolkit [2]



[2] Z. Huang et al. RIOT-ML: Toolkit for Over-the-Air Secure Updates and Performance Evaluation of TinyML Models. In Annals of Telecommunications, 2024

Implementation Details

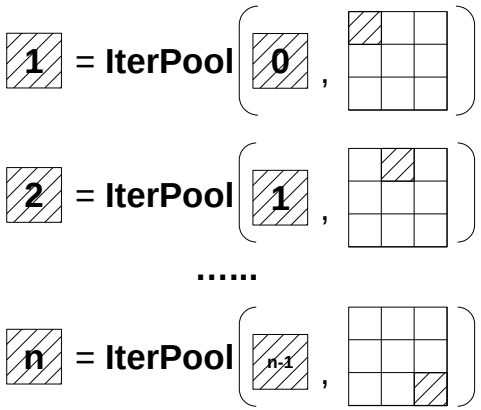
Common Global Pooling



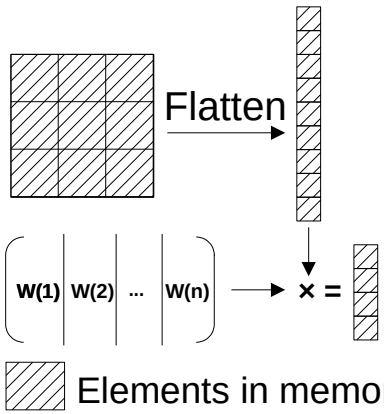
(Iter)Pool ∈ {min, max, average}

Elements in memory

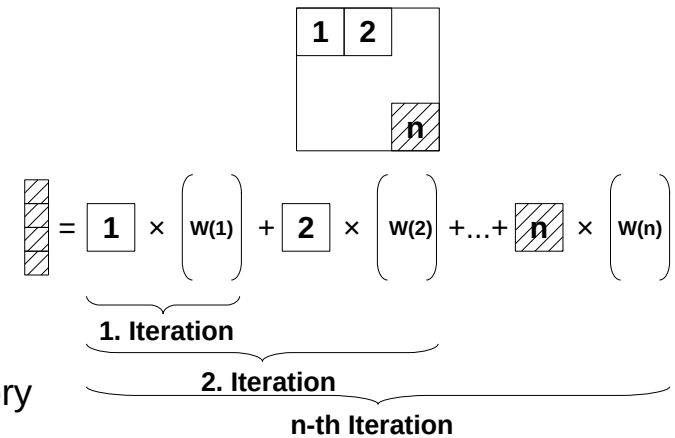
Iterative Global Pooling



Common Dense



Iterative Dense



Agenda

- (Tiny) Machine Learning
- Challenges in TinyML
- Problem Statement
- Encoding Costs in DAG
- Implementation Details
- Experiments on MCUs
- Conclusion
- On-going Work

Experiments on MCUs

Tabelle: The different MCUs & boards used in our experiments. The RAM and Flash capacity are presented in kB.

Board	MCU Core	RAM	Flash
Nucleo-f767zi	Cortex-M7 @ 216 MHz	512	2048
Stm32f746g-disco	Cortex-M7 @ 216 MHz	320	1024
Nucleo-f412zg	Cortex-M4 @ 100 MHz	256	1024
esp32s3-devkit	Xtensa @ 240 MHz	512	8192
esp32c3-devkit	RISC-V @ 160 MHz	384	4096
hifive1b	RISC-V @ 320 MHz	16	4096

Experiments on MCUs

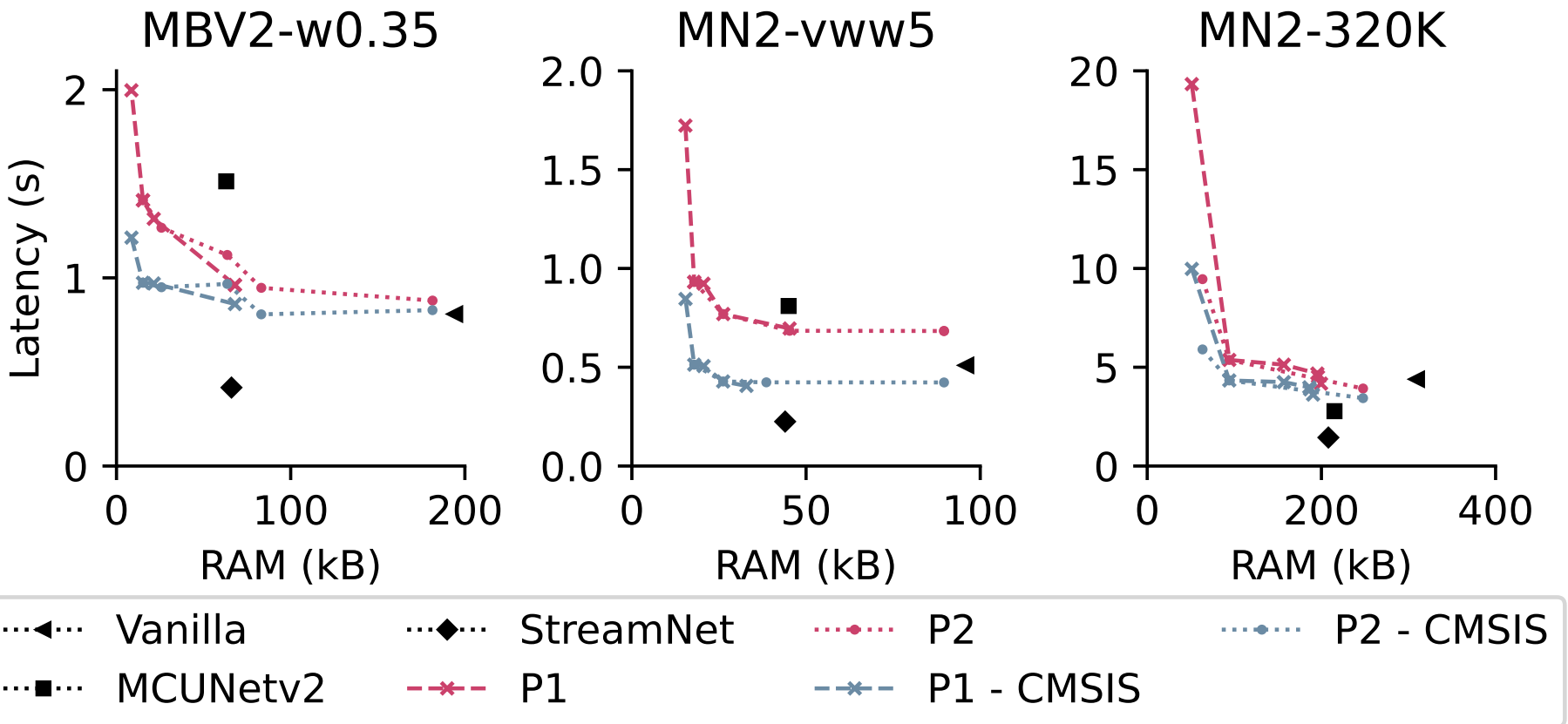
msf-CNN achieves the lowest peak RAM usage.

Tabelle: 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

Experiments on MCUs

msf-CNN allows trade-off between peak RAM usage and compute cost while applying patch-based fusion.



Vanilla: no Patch-based fusion applied.
Benchmarks on stm32f767 MCU (ARM Cortex-M7).

Agenda

- (Tiny) Machine Learning
- Challenges in TinyML
- Problem Statement
- Encoding Costs in DAG
- Implementation Details
- Experiments on MCUs
- Conclusion
- On-going Work

Conclusion

In a nutshell

- msf-CNN is a new fusion technique for ultra-low RAM
- Open source implementation
- Evaluation on heterogeneous MCU-based IoT hardware
- Enables new RAM/latency trade-offs

Conclusion

Remarks:

- Reminder: Precision is not affected (the model architecture and weights are untouched).
- Orthogonal techniques decreasing model size: quantization, model pruning... these can be used alongside.
- With appropriate cost models and backends, msf-CNN can be extended to accelerators (GPUs, FPGAs, NPUs).

Future Work

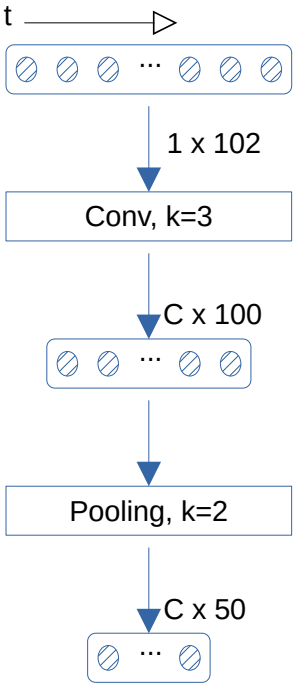
- Expand the parameter space and cost model: Patch size, power usage etc.
- Integrate alternative cache paradigm, like 2D cache in StreamNet.
- Support more network architectures: Transformer, RNN/LSTM etc.

Agenda

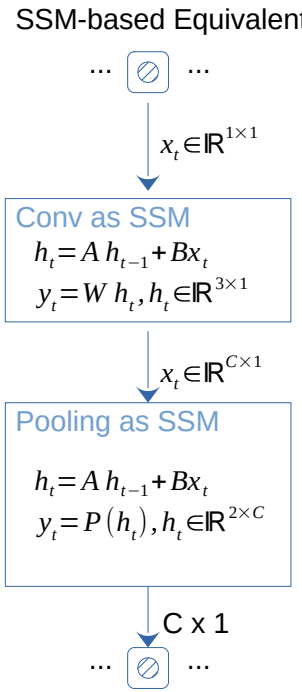
- (Tiny) Machine Learning
- Challenges in TinyML
- Problem Statement
- Encoding Costs in DAG
- Implementation Details
- Experiments on MCUs
- Conclusion
- On-going Work

TinyDéjàVu: Smaller RAM and Faster Streaming Inference

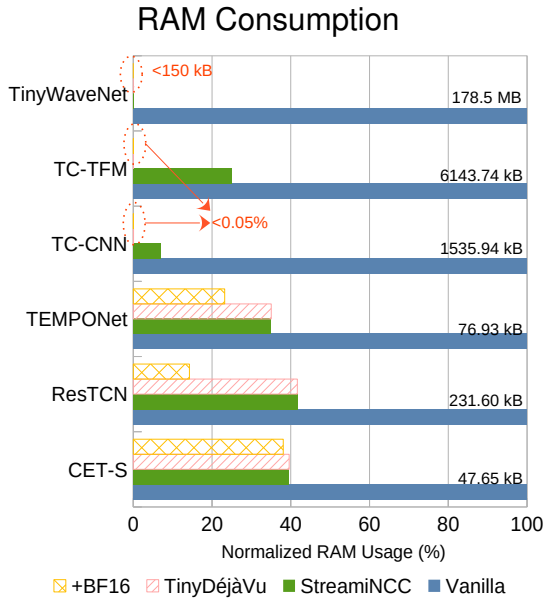
Idea: Replace temporal operators with State-Space Models (SSMs).

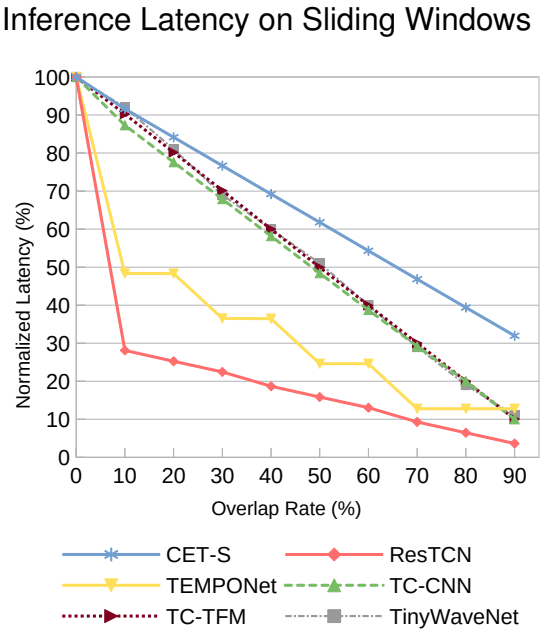


Peak RAM = C x 150



Peak RAM = 3 + 4C
(0.02%)

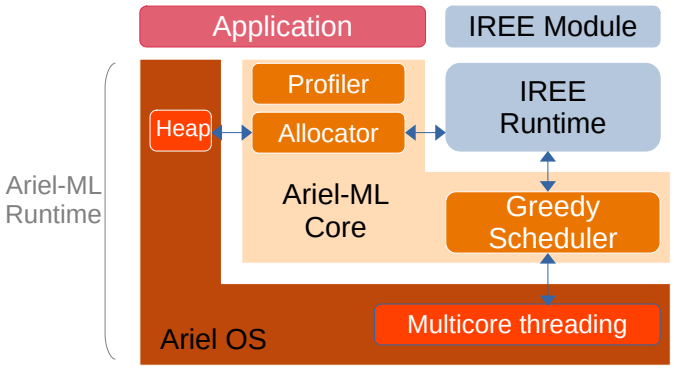




- Save up to 90 % of RAM usage compared to StreamiNCC (SOTA).
- Drastically reduce inference latency on overlap sliding windows.
- No loss of task-level performance.

Ariel-ML: Inference Engine for Multicore MCUs

Build upon Ariel OS and IREE.



Inference execution time (in ms) for quantized LeNet5 and MCUNet.

MCU	RIOT-ML	RIOT+IREE	Ariel-ML
(LeNet5)			
nRF52840	66.088	64.573	63.721
ESP32-C3	54.953	42.138	44.17
RP2040	70.117	50.557	46.757
RP2040+multicore	(NA)	(NA)	31.543 (1.5×)
RP2350	(NS)	(NS)	27.966
RP2350+multicore	(NS)	(NS)	19.630 (1.4×)
(MCUNet)			
nRF52840	1682.124	990.638	981.870
RP2040	1751.570	1055.623	1057.223
RP2040+multicore	(NA)	(NA)	661.530 (1.6×)
RP2350	(NS)	(NS)	396.478
RP2350+multicore	(NS)	(NS)	280.109 (1.4×)

(NA: not applicable; NS: Not supported by OS)

- First engine to natively support multi-core capabilities on microcontrollers.
- Up to a 1.6× speedup with multicore enabled, approaching the hardware's known limit imposed by bus contention.
- Rust-based, strong memory-safety guarantee.

Thanks! And Questions?

Code:

https://github.com/TinyPART/msf-CNN
https://github.com/TinyPART/TinyChirp
https://github.com/ariel-os/ariel-ml

Papers:

NeurIPS 2025 msf-CNN https://arxiv.org/pdf/2505.11483
IEEE IoS 2024 TinyChirp https://arxiv.org/pdf/2407.21453
Tiny*DéjàVu*: https://arxiv.org/pdf/2512.09786
Ariel-ML: https://arxiv.org/pdf/2512.09800

E-Mail: zhaolan.huang@fu-berlin.de