tinyML® EMEA

Enabling Ultra-low Power Machine Learning at the Edge

June 26 - 28, 2023

www.tinyML.org

muRISCV-NN: Deep-Learning Inference Kernels for Embedded Platforms using the RISC-V Vector and Packed Extensions

Philipp van Kempen, Fabian Peddinghaus,

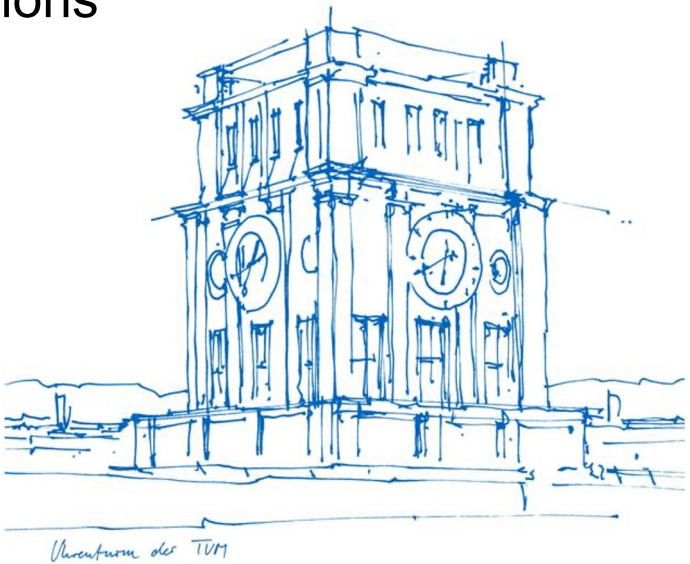
Jefferson Parker Jones, Rafael Stahl,

Daniel Müller-Gritschneder, Ulf Schlichtmann

Technical University of Munich

TUM School of Computation, Information and Technology

Chair for Electronic Design Automation



27.06.2023

This work has been developed in the ZuSE project Scale4Edge. Scale4Edge is funded by the German ministry of education and research (BMBF) (reference numbers: 16ME0122K-16ME0140+16ME0465). The authors are responsible for the content of this publication.

SPONSORED BY THE

Federal Ministry of Education and Research

Agenda

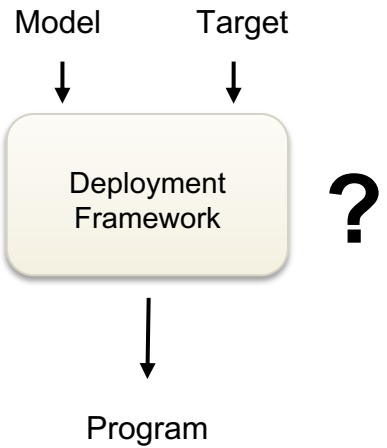
1. **Motivation**
   - TinyML Workloads
   - TinyML Targets
   - TinyML Deployment
   - Edge ML Pipeline
2. **Methodology**
   - Implementation
   - Supported Operators
   - Infrastructure
3. **Evaluation**
   - MLPerf Tiny Benchmark
   - Performance Results
   - Overheads
4. **Conclusion**
   - Challenges
   - Outlook

Agenda

1. **Motivation**
   - TinyML Workloads
   - TinyML Targets
   - TinyML Deployment
   - Edge ML Pipeline
2. **Methodology**
   - Implementation
   - Supported Operators
   - Infrastructure
3. **Evaluation**
   - MLPerf Tiny Benchmark
   - Performance Results
   - Overheads
4. **Conclusion**
   - Challenges
   - Outlook

Motivation

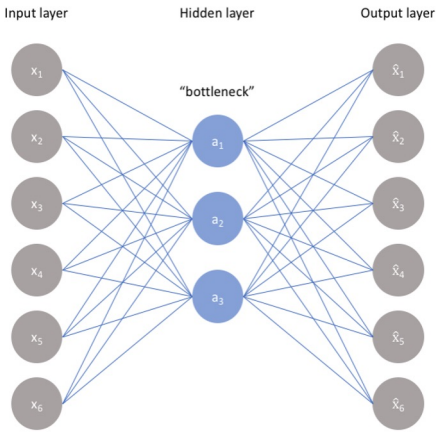
- TinyML Workloads and targets are emerging rapidly
- RISC-V ISA becoming more popular
- Choosing the right ML Deployment Framework is non-trivial



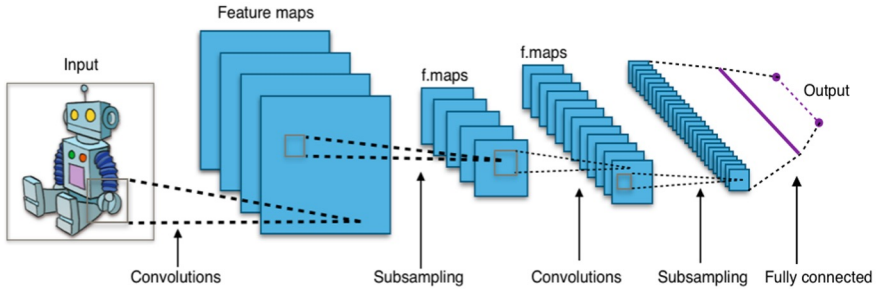
Here:
- **Quantized Networks only**
- **Inference only**

TinyML Workloads

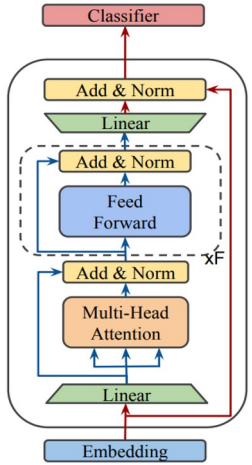
Deep AutoEncoder*



Convolutional Neural Network (CNN)*



Transformer

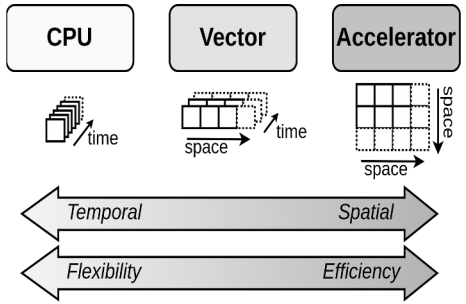


*MLPerf™ Tiny workloads

TinyML Targets

Architectures for Embedded ML:

- CPU
- Vector
- Accelerators



RISC-V Packed Extension

- Still in development (v0.9.6)
- Sub-word SIMD (i8v4, i16v2,...)

RISC-V Vector Extension (RVV)

- Ratified in Nov. 2021 (v1.0)
- Super-word SIMD
- Embedded Vector Zve32x for MCUs

Here:

- RISC-V MCU with optional SIMD support
  - Scalar: RV32IM[FD]C → comparable to Cortex-M0
  - Packed: RV32IM[FD]C**P** → comparable to Cortex-M4 (DSP)
  - Vector: RV32IM[FD]C**V** →comparable to Cortex-M55 (DSP+MVEI/Helium)

RISC-V Vector Cores

Academic

- **Hwacha** - UC Berkeley
- **Ara** - ETH Zurich
- **Vitruvius+** - BSC
- **RISC-V²** - Univ. of Thrace
- …

Commercial

- **OpenC906 / OpenC910** - Alibaba
- **P270 / X280** - SiFive
- **NX27V** - Andes
- **NS-72 / DR1000C** - NSITEXE
- …

Embedded **Zve32x**

- 2021 - **Vicuna** [1] - TU Wien
- 2022 - **Spatz** - ETH Zurich
- …

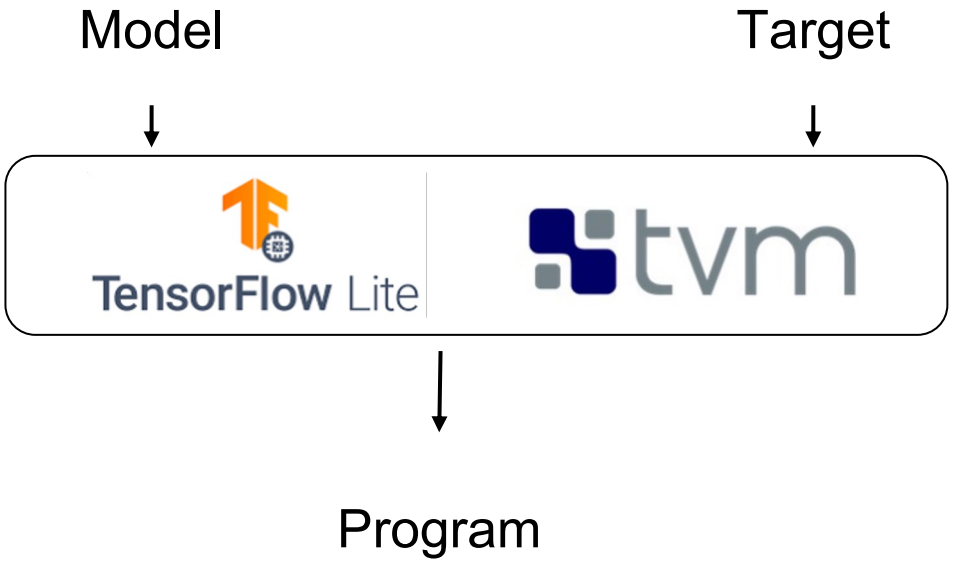
TinyML Deployment

Frameworks

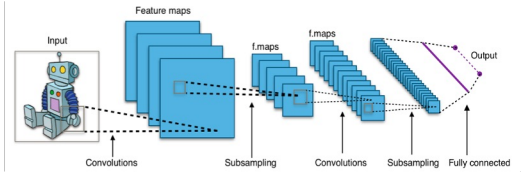
- TFLite for Microcontrollers [2] (TFLM) 
  - Industry standard
  - Many vendor libraries
  - Hardcoded kernel → Generic or hand-tuned

- TVM [3] 
  - Follows compiler-driven approach → Allows many optimizations & auto-tuning
  - Highly interesting research field
  - MicroTVM: Deployment of TVM programs to MCUs
  - Provides different ways to integrate accelerators: BYOC/UMA

Edge ML Pipeline



Edge ML Pipeline



Target

TensorFlow Lite

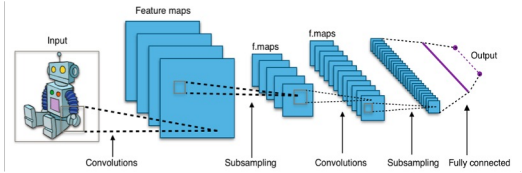
tvm

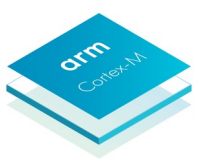
Program

Edge ML Pipeline

ARM:
- Scalar
- DSP
- Helium

RISC-V:
- Scalar
- Packed
- Vector





TensorFlow Lite

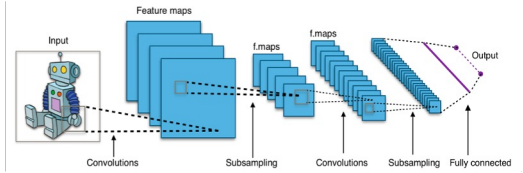
tvm

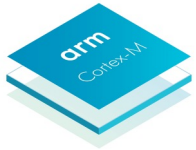
Program

Edge ML Pipeline

ARM:
- **Scalar**
- DSP
- Helium

RISC-V:
- **Scalar**
- Packed
- Vector





TensorFlow Lite

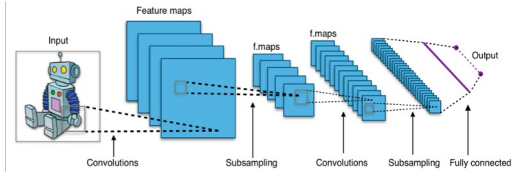
tvm

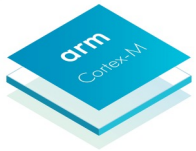
Generic C/C++

Edge ML Pipeline

ARM:
- **Scalar**
- **DSP**
- **Helium**

RISC-V:
- **Scalar**
- Packed
- Vector





TensorFlow Lite

tvm

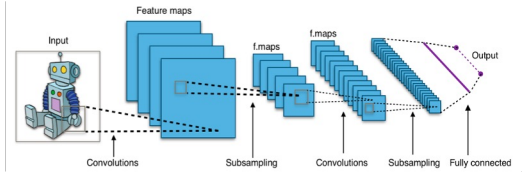
CMSIS

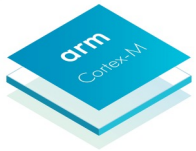
Provided by ARM

Edge ML Pipeline

ARM:
- Scalar
- DSP
- Helium

RISC-V:
- Scalar
- Packed
- Vector





TensorFlow Lite

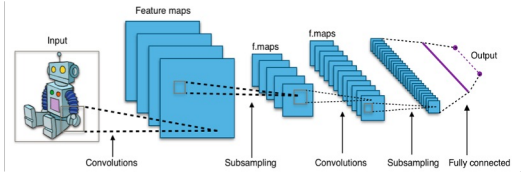
tvm

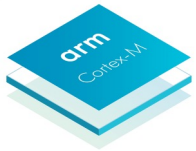
muRISCV-NN

Edge ML Pipeline

ARM:
- Scalar
- DSP
- Helium

RISC-V:
- Scalar
- Packed
- Vector





TensorFlow Lite

tvm

CMSIS

Generic C/C++

muRISCV-NN

Agenda

1. **Motivation**
   - TinyML Workloads
   - TinyML Targets
   - TinyML Deployment
   - Edge ML Pipeline

2. **Methodology**
   - Implementation
   - Supported Operators
   - Infrastructure

3. **Evaluation**
   - MLPerf Tiny Benchmark
   - Performance Results
   - Overheads

4. **Conclusion**
   - Challenges
   - Outlook

Methodology

Design Idea

- Implement a kernel library for efficient NN inference on RISC-V MCUs

Approach

- Use ARM CMSIS-NN [4] as baseline implementation
- Keep interface consistent for compatibility with 3rd party tools

Implementation

- Supported modes:
  - Scalar (portable): unchanged
  - Vector (RVV): Implemented & hand-optimized
  - Packed (RVP): Implemented & hand-optimized
- Using C-level intrinsic functions instead of Inline-Assembly!

Supported Operators & Types

NN Operators:

- (Depthwise Separable) Convolution
- Fully Connected
- Softmax
- ReLU
- Max Pooling / Average Pooling
- Elementwise Addition / Multiplication
- Singular Value Decomposition
- New: LSTM

Data Types:

- int8/int16 only

Only Quantized Networks → QNNs

Inference only → No on-device training

Infrastructure

Unit Tests:

- Passing the same unit tests as CMSIS-NN
  → bit exact!

Integration Tests:

- TFLite Micro (TFLM)
  - Can be used as a drop-in replacement for CMSIS-NN by applying a minimal patch
- MicroTVM
  - Reusing existing CMSIS-NN BYOC (Bring-Your-Own-Codegen) integration
- MLonMCU
  - TinyML deployment tool used for benchmarking

CI/CD Flow:

- Code Style Checks
- Weekly builds + tests
- Automated Benchmarks

Miscellaneous :

- Providing instructions for obtaining toolchains (LLVM/GCC) and simulators
- Extensive user/developer documentation

Agenda

1. **Motivation**
   - TinyML Workloads
   - TinyML Targets
   - TinyML Deployment
   - Edge ML Pipeline
2. **Methodology**
   - Implementation
   - Supported Operators
   - Infrastructure
3. **Evaluation**
   - MLPerf Tiny Benchmark
   - Performance Results
   - Overheads
4. **Conclusion**
   - Challenges
   - Outlook

Benchmarks

MLPerf Tiny Benchmarking Suite [5]

Deep Learning Benchmarks for Embedded Devices

Name	Use Case	Model
aww	Keyword Spotting	DS-CNN
vww	Visual Wake Words	MobileNet
resnet	Image Classification	ResNet
toycar	Anomaly Detection	Deep AutoEncoder

Setup

Simulator

- Spike Instruction Set Simulator
- Support various RISC-V extensions, including V(ector) and P(acked)
- Also known as `riscv-isa-sim`

Toolchain

- LLVM 14: Scalar + Vector
- GCC (Draft): Packed

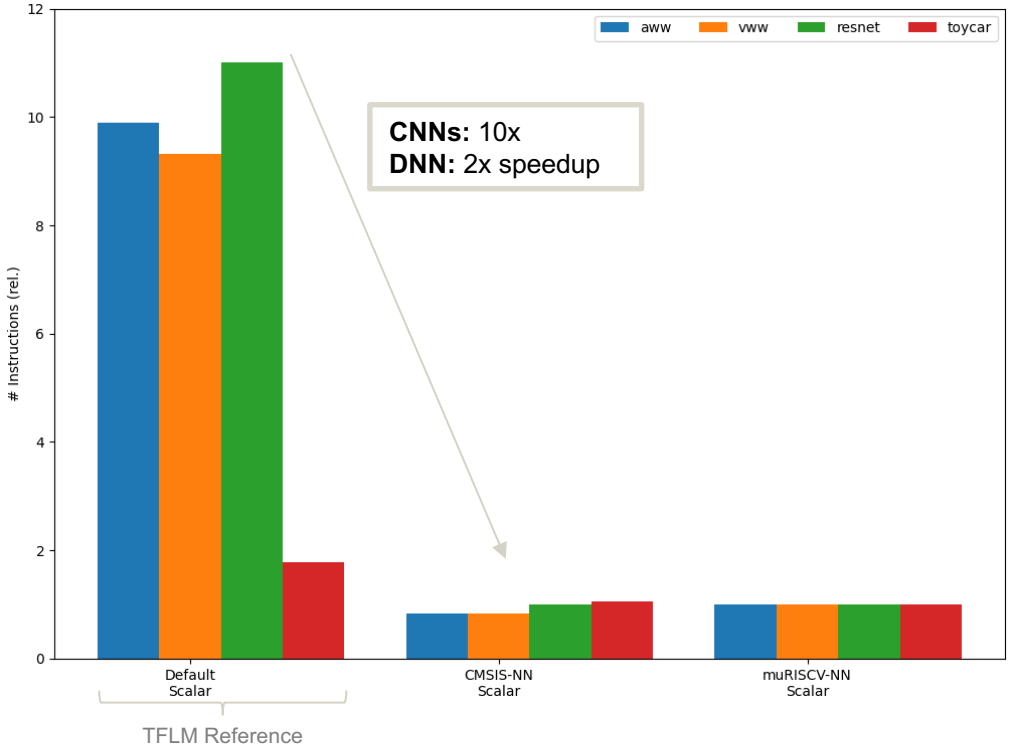
Flow

- All benchmarks generated with MLonMCU[6] Tiny Deployment tool

Results: muRISCV-NN vs. CMSIS-NN vs. TFLM

Default:
TFLM Reference Kernels
→ Not optimized

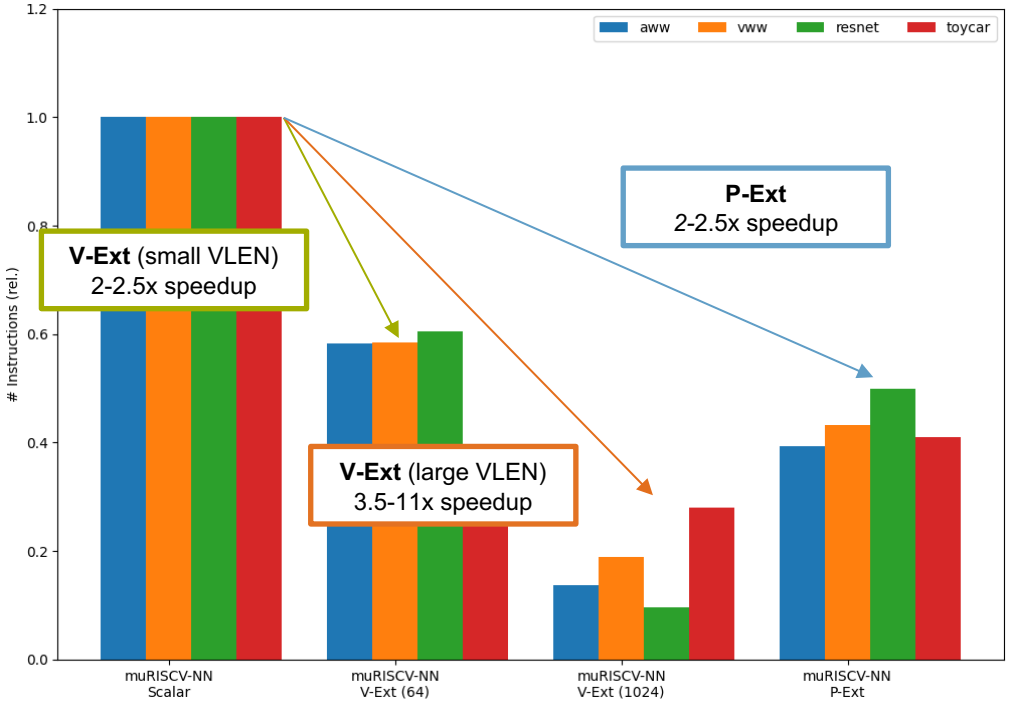
CMSIS-NN & muRISCV-NN:
Only ±10% differences



Results: Scalar vs. Vector vs. Packed

V-Ext:
Runtime falls with higher VLEN up to 2048 bits.

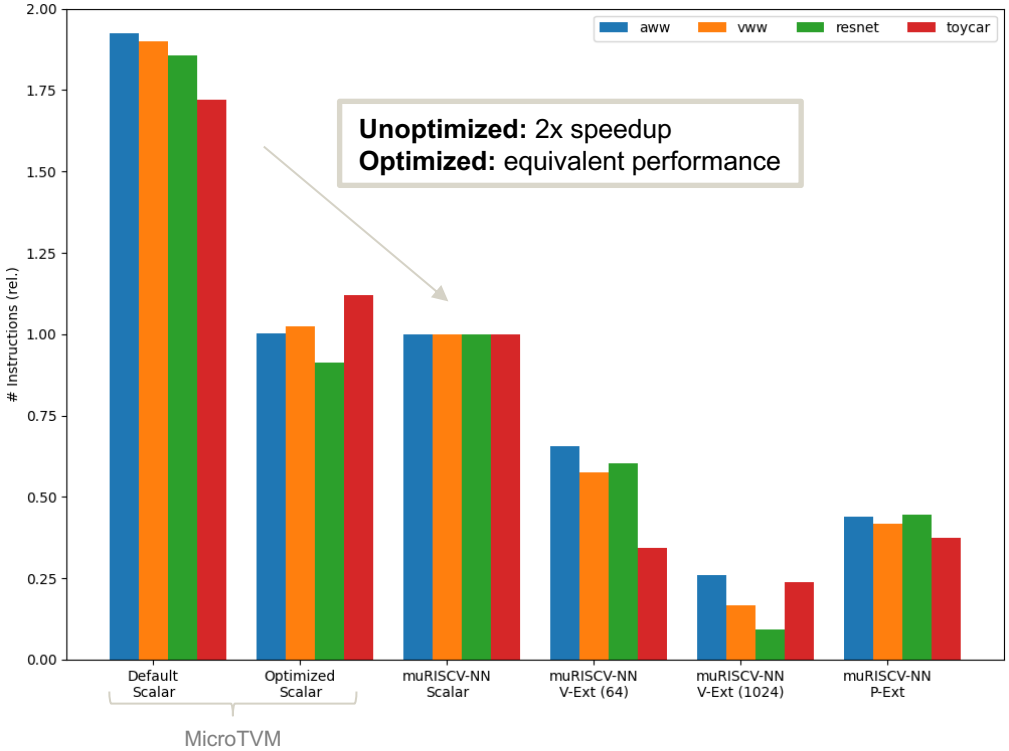
P-Ext:
Less overhead for configuration instructions and load/stores



Results: muRISCV-NN vs. TVM

Default:
NHWC Layout
Fallback Schedules

Optimized:
NCHW Layout
→ transformed
Tuned Schedules
→ > 2h per Model



Memory overheads

ROM (constants + code size)

- muRISCV-NN requires ~5% more ROM compared to TFLM reference implementation
- P-Ext kernel implementations require 5-10% more ROM

RAM (intermediate buffers)

- No extra overhead vs. TFLM default implementations
- Between 1.5-3.1 times less RAM compared to TVM kernels

Basically equivalent to CMSIS-NN memory metrics!

Agenda

1. **Motivation**
   - TinyML Workloads
   - TinyML Targets
   - TinyML Deployment
   - Edge ML Pipeline
2. **Methodology**
   - Implementation
   - Supported Operators
   - Infrastructure
3. **Evaluation**
   - MLPerf Tiny Benchmark
   - Performance Results
   - Overheads
4. **Conclusion**
   - Challenges
   - Outlook

Challenges

Vector Extension (RVV)

- Can not utilize full vector length due to limited channel lengths in some convolutions
- Layout transformations not possible with CMSIS-NN/muRISCV-NN (NHWC only)

Packed Extension (RVP)

- Unable to achieve full potential due to TFLM quantization scheme (8bit + offset) → 16-bit inputs

Workarounds

- RISC-V does not support ARM rounding mode → Emulation is costly!

Maintenance

- Staying in-sync with upstream progress

Outlook

Work In Progress

- Optimization of kernels (specially P-Extension and special cases)
- Support more (academic) cores

Future Work

- Support hardware targets
- Alternative Rounding Modes

muRISCV-NN

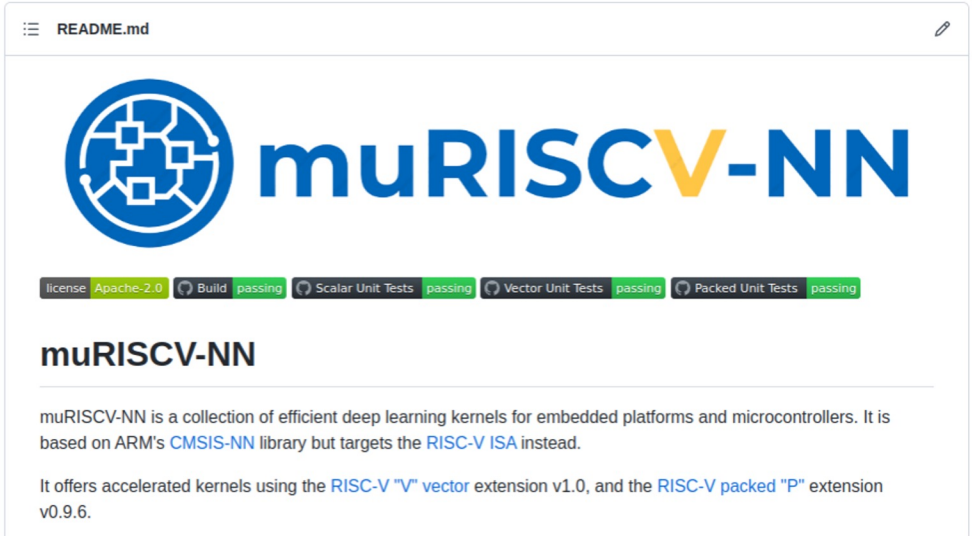
Efficient deep learning kernels for RISC-V microcontrollers.

- CMSIS-NN fork
  - Same unit tests
  - ⇾ **functionally equivalent!**
- RISC-V extensions
  - **P packed** 0.9.6
  - **V vector** 1.0 (Zve32x)
- Supporting
  - **ISS:** Spike, riscvOVPsim, ETISS
  - **RTL:** Vicuna
  - **HW:** soon!

- Toolchains
  - **RISC-V GCC** & **LLVM 14+**
- Integration with
  - **TensorFlow Lite**
  - **TVM**
- Performance
  - Scalar — Up to 10x faster than TFLM; On par with tuned TVM
  - Packed — Up to 2.5x faster than Scalar
  - Vector — Up to 11x faster than Scalar

muRISCV-NN

Efficient deep learning kernels for RISC-V microcontrollers.


README.md

muRISCV-NN

license Apache-2.0 | Build passing | Scalar Unit Tests passing | Vector Unit Tests passing | Packed Unit Tests passing

muRISCV-NN

muRISCV-NN is a collection of efficient deep learning kernels for embedded platforms and microcontrollers. It is based on ARM's CMSIS-NN library but targets the RISC-V ISA instead.

It offers accelerated kernels using the RISC-V "V" vector extension v1.0, and the RISC-V packed "P" extension v0.9.6.

Open Source

https://github.com/tum-ei-eda/muriscv-nn

References

[1] Platzer, M., & Puschner, P. (2021). Vicuna: a timing-predictable RISC-V vector coprocessor for scalable parallel computation. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[2] David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., ... & Rhodes, R. (2021). Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems*, *3*, 800-811.

[3] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., ... & Krishnamurthy, A. (2018). TVM: An automated end-to-end optimizing compiler for deep learning. *arXiv preprint arXiv:1802.04799*.

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

[5] Banbury, C., Reddi, V. J., Torelli, P., Holleman, J., Jeffries, N., Kiraly, C., ... & Xuesong, X. (2021). Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597*.

[6] van Kempen, P., Stahl, R., Mueller-Gritschneder, D., & Schlichtmann, U. (2023). MLonMCU: TinyML benchmarking with fast retargeting. *arXiv preprint arXiv:2306.08951.*

Questions

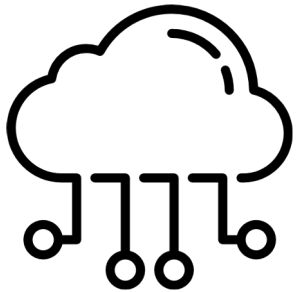
Bonus slides

Why ML at the Edge?

ML needs **Computing power & Memory bandwidth**

Offload computation to the **cloud?**

- ▲ Plenty of resources available
- ▼ High **latency**
- ▼ Low **bandwidth**
- ▼ Poor **reliability**
- ▼ **Privacy** concerns

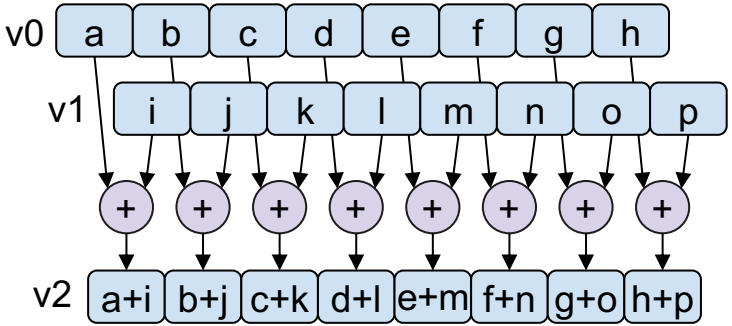


➡️ **Cloud is not always a good choice!**

Why Vectors? Why not SIMD?

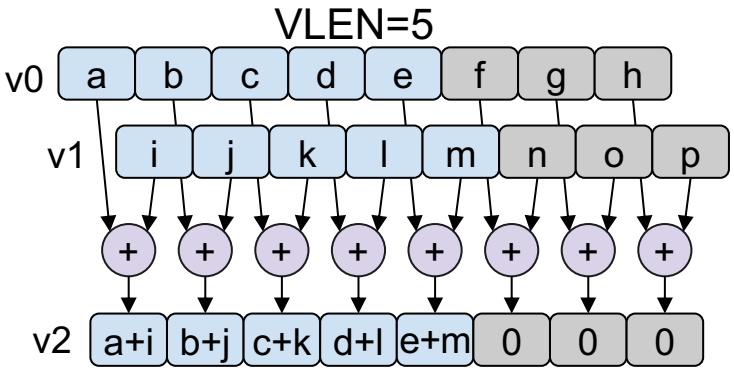
SIMD

- SIMD has **fixed length** ☹
- Software **hard to maintain** ☹



Vectors

- Vector machines are **length-agnostic** ☺
- One software **fits all** ☺



Why Vectors? Why not SIMD?

```
void daxpy(size_t n, double a, const double x[], double y[])
{
 for (size_t i = 0; i < n; i++) {
   y[i] = a*x[i] + y[i];
 }
}
```

MIPS (MSA)

```
# a0 is n, a2 is pointer to x[0], a3 is pointer to y[0], $w13 is a
  0:  li      a1,-2
  4:  and     a1,a0,a1          # a1 = floor(n/2)*2 (mask bit 0)
  8:  sll     t0,a1,0x3         # t0 = byte address of a1
  c:  addu    v1,a3,t0          # v1 = &y[a1]
 10:  beq     a3,v1,38          # if y==&y[a1] goto Fringe
                                #    (t0==0 so n is 0 | 1)
 14:  move    v0,a2             # (delay slot) v0 = &x[0]
 18:  splati.d $w2,$w13[0]      # w2 = fill SIMD reg. with copies of a
Main Loop:
 1c:  ld.d    $w0,0(a3)         # w0 = 2 elements of y
 20:  addiu   a3,a3,16          # incr. pointer to y by 2 FP numbers
 24:  ld.d    $w1,0(v0)         # w1 = 2 elements of x
 28:  addiu   v0,v0,16          # incr. pointer to x by 2 FP numbers
 2c:  fmadd.d $w0,$w1,$w2       # w0 = w0 + w1 * w2
 30:  bne     v1,a3,1c          # if (end of y != ptr to y) go to Loop
 34:  st.d    $w0,-16(a3)       # (delay slot) store 2 elts of y
Fringe:
 38:  beq     a1,a0,50          # if (n is even) goto Done
 3c:  addu    a2,a2,t0          # (delay slot) a2 = &x[n-1]
 40:  ldc1    $f1,0(v1)         # f1 = y[n-1]
 44:  ldc1    $f0,0(a2)         # f0 = x[n-1]
 48:  madd.d  $f13,$f1,$f13,$f0# f13 = f1+f0*f13 (muladd if n is odd)
 4c:  sdc1    $f13,0(v1)        # y[n-1] = f13 (store odd result)
Done:
 50:  jr      ra                # return
 54:  nop                       # (delay slot)
```

Intel (AVX2)

```
# eax is i, n is esi, a is xmm1,
# pointer to x[0] is ebx, pointer to y[0] is ec
 0:  push   esi
 1:  push   ebx
 2:  mov    esi,[esp+0xc]   # esi = n
 6:  mov    ebx,[esp+0x18]  # ebx = x
 a:  vmovsd xmm1,[esp+0x10] # xmm1 = a
 10:  mov    ecx,[esp+0x1c]  # ecx = y
 14:  vmovddup xmm2,xmm1    # xmm2 = {a,a}
 18:  mov    eax,esi
 1a:  and    eax,0xfffffffc  # eax = floor(n/4)*4
 1d:  vinsertf128 ymm2,ymm2,xmm2,0x1 # ymm2 = {a,a,a,a}
 23:  je     3e              # if n < 4 goto Fringe
 25:  xor    edx,edx         # edx = 0
Main Loop:
 27:  vmovapd ymm0,[ebx+edx*8] # load 4 elements of x
 2c:  vfmadd213pd ymm0,ymm2,[ecx+edx*8] # 4 mul adds
 32:  vmovapd [ecx+edx*8],ymm0 # store into 4 elements of y
 37:  add    edx,0x4
 3a:  cmp    edx,eax         # compare to n
 3c:  jb     27              # repeat loop if < n
Fringe:
 3e:  cmp    esi,eax         # any fringe elements?
 40:  jbe    59              # if (n mod 4) == 0 go to Done
Fringe Loop:
 42:  vmovsd xmm0,[ebx+eax*8] # load element of x
 47:  vfmadd213sd xmm0,xmm1,[ecx+eax*8] # 1 mul add
 4d:  vmovsd [ecx+eax*8],xmm0 # store into element of y
 52:  add    eax,0x1         # increment Fringe count
 55:  cmp    esi,eax         # compare Loop and Fringe counts
 57:  jne    42 <daxpy+0x42>  # repeat FringeLoop if != 0
Done:
 59:  pop    ebx             # function epilogue
 5a:  pop    esi
 5b:  ret
```

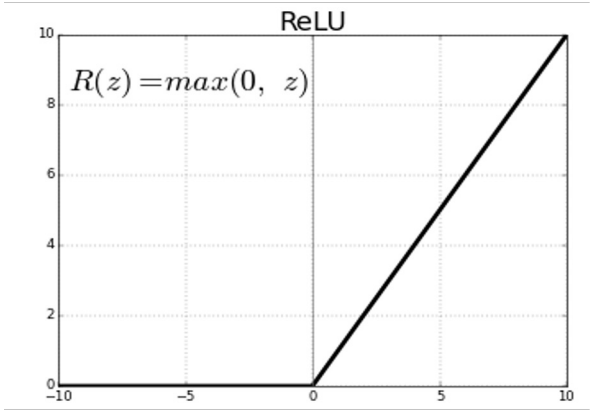
RISC-V (RVV)

```
# a0 is n, a1 is pointer to x[0], a2 is pointer to y[0], fa0 is a
 0:  li t0, 2<<25
 4:  vsetdcfg t0             # enable 2 64b Fl.Pt. registers
loop:
 8:  setvl  t0, a0           # vl = t0 = min(mvl, n)
 c:  vld    v0, a1           # load vector x
 10:  slli   t1, t0, 3        # t1 = vl * 8 (in bytes)
 14:  vld    v1, a2           # load vector y
 18:  add    a1, a1, t1       # increment pointer to x by vl*8
 1c:  vfmadd v1, v0, fa0, v1  # v1 += v0 * fa0 (y = a * x + y)
 20:  sub    a0, a0, t0       # n -= vl (t0)
 24:  vst    v1, a2           # store Y
 28:  add    a2, a2, t1       # increment pointer to y by vl*8
 2c:  bnez   a0, loop         # repeat if n != 0
 30:  ret                     # return
```

RISC-V V Extension: ReLU Example

Reference in plain C

```
for (uint16_t i = 0; i < size; i++) {
    if (data[i] < 0) data[i] = 0;
}
```



RISC-V Vector

```
# a0=data, a1=size
for:
   vsetvli t0, a1, e8, m8, ta, ma   # Vectors of 8bit
   vle8.v v0, (a0)                  # Load bytes
   vmax.vx v0, v0, zero             # Apply activation
   vse8.v v0, (a0)                  # Store bytes
   add a0, a0, t0                   # Decrement size
   sub a1, a1, t0                   # Bump pointer
   bnez a1, for                     # Any more?
```

Copyright Notice

www.tinyml.org