Carnegie Mellon University
Software Engineering Institute

PHITE:
Portable High-performance Inference at the Tactical Edge

NOVEMBER 14, 2022

Scott McMillan
Principal Research Engineer

Carnegie Mellon University Software Engineering Institute

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-0873

PHITE: Enabling AI for Decision-Making Advantage at the Tactical Edge

Problem: Today's AI software is computationally expensive and requires extensive knowledge, skill, and effort to adopt on low-power devices at the tactical edge.

Solution: Develop an open-source library of machine learning (ML) algorithms optimized for low-power (100's mW—0's W) embedded devices.

DoD Benefit

- Aid deployment of ML across a spectrum of edge-based applications.
- Enable rapid adoption of new/novel embedded hardware architectures.
- Provide efficient use of limited hardware for performance gains in AI/ML applications.
- Enable a wider range of applications at the tactical edge through portable and more capable software foundations.

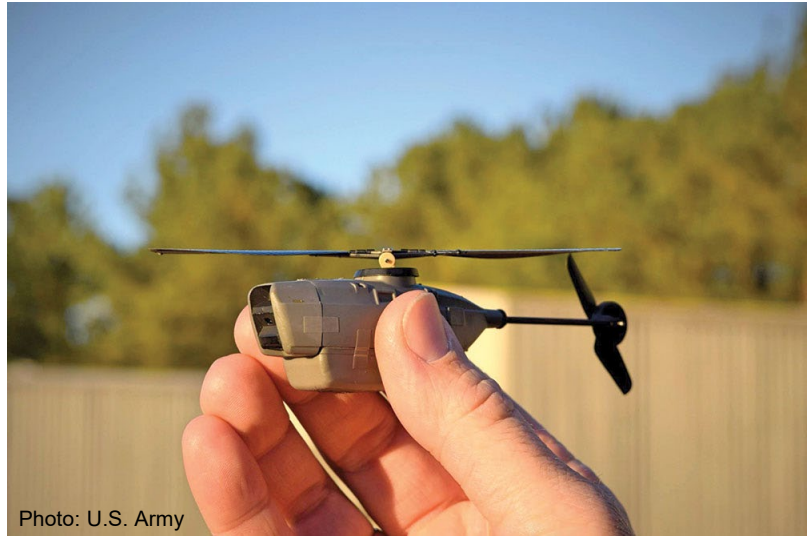


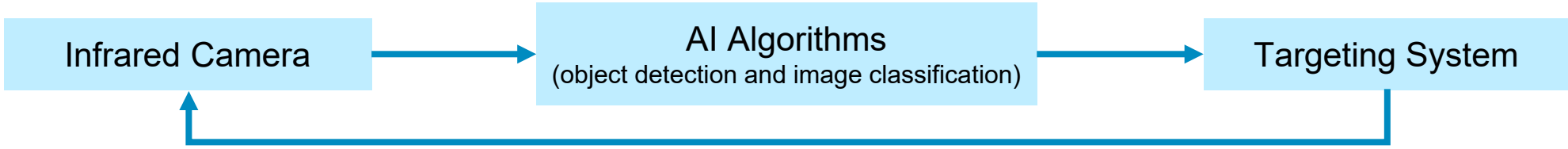
Photo: U.S. Army

Areas of Opportunity

- Soldier-borne sensors
- Predictive maintenance
- Unattended sensors
- IoT/Io(B)T devices

Longer operational times → Increased situational awareness/force protection • Less weight →Increased mobility

ATLAS Initiative



The Advanced Targeting and Lethality Aided System (ATLAS) is an **emerging targeting technology** being developed by DEVCOM's C5ISR and Armaments Centers. ATLAS uses cutting-edge sensing technologies and **machine-learning algorithms** to **automate** manual tasks during **passive target acquisition**, allowing crews to engage three targets in the time it would normally take for them to engage one.

"Advancements from the PHITE project will improve mission critical parameters in current edge systems and make possible new edge systems."

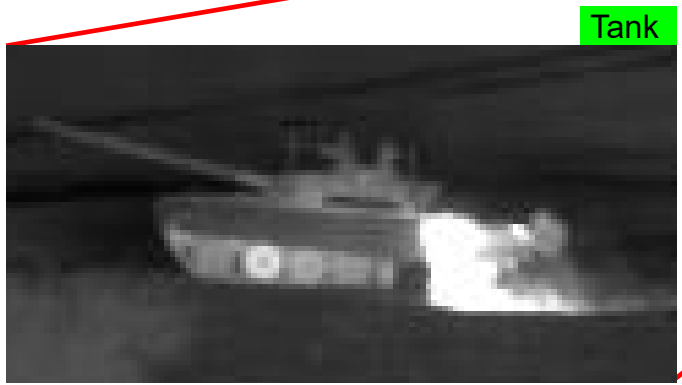
— Forrest Bussler, Chief, Embedded Hardware and Processing Branch, US Army DEVCOM C5ISR Center

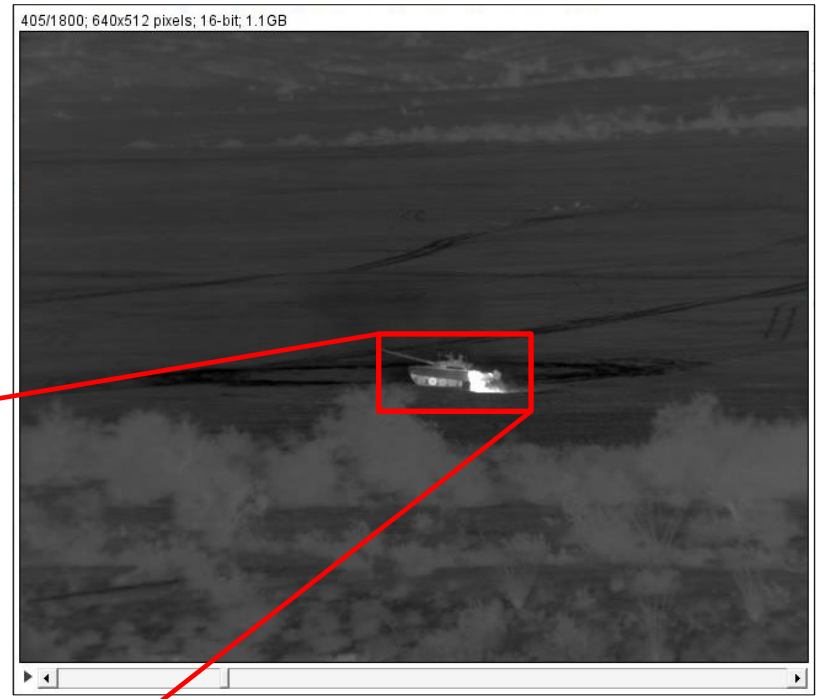
Photo: U.S. Army

Dataset: Automatic Target Recognition (ATR)

300 GB of full-motion video clips at multiple ranges and aspects (moving in circles, walking in figure eights):

- Tanks
- Armored vehicles
- Trucks
- People





https://dsiac.org/databases/atr-algorithm-development-image-database/

Our goal is to maximize analytic capability **at the smallest scales.**

System:	DGX-2	Jetson AGX Xavier	Raspberry Pi PICO Microcontroller
Power	10kW	<30W	330mW
Cost	$399,000	$999	$4
Memory	1TB system/512GB GPU	32GB 256-bit LPDDR4x	264KB RAM (2MB flash)
Processors	Intel Platinum (24 cores) x 2 + NVIDIA Tesla V100 x 16	512-core Volta GPU w/ 64 Tensor Cores (8 Volta SMs)	RP2040: ARM Cortex-M0+ (dual core)
Peak	2 petaFLOPS	1.41 teraFLOPS	266 megaFLOPS
Model/Size	BiT-M(ResNet) / 900M parameters	AlexNet / 60M parameters	MobileNet V2 / 3M parameters

Push analytics capability to the right.

Approach: Extend and Apply CMU's Research on Direct Convolutions

High Performance Zero-Memory Overhead Direct Convolutions

Jiyuan Zhang [1] Franz Franchetti [1] Tze Meng Low [1]

Abstract

The computation of convolution layers in deep neural networks typically rely on high performance routines that trade space for time by using additional memory (either for packing purposes or required as part of the algorithm) to improve performance. The problems with such an approach are two-fold. First, these routines incur additional memory overhead which reduces the overall size of the network that can fit on embedded devices with limited memory capacity. Second, these high performance routines were not optimized for performing convolution, which means that the performance obtained is usually less than conventionally expected. In this paper, we demonstrate that direct convolution, when implemented *correctly*, eliminates all memory overhead, and yields performance that is between 10% to 400% times

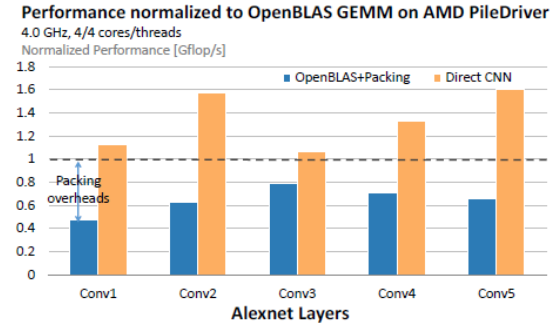


Figure 1. High performance direct convolution implementation achieves higher performance than a high performance matrix multiplication routine, whereas matrix-multiplication based convolution implementations suffers from packing overheads and is limited by the performance of the matrix multiplication routine

In *International Conference on Machine Learning*, pp. 5776-5785. PMLR, 2018.

Our Team

Carnegie Mellon University Software Engineering Institute

CMU / SEI

Dr. Scott McMillan, Principal Investigator
Principal Engineer – MTS,
AI Division, SEI

Jay Palat
Senior Engineer,
AI Division, SEI

Oren Wright
Senior Researcher – MTS,
AI Division, SEI

CMU / ECE

Prof. Tze Meng Low, Co-Principal Investigator
Assistant Research Professor, Electrical and Computer Engineering, CMU

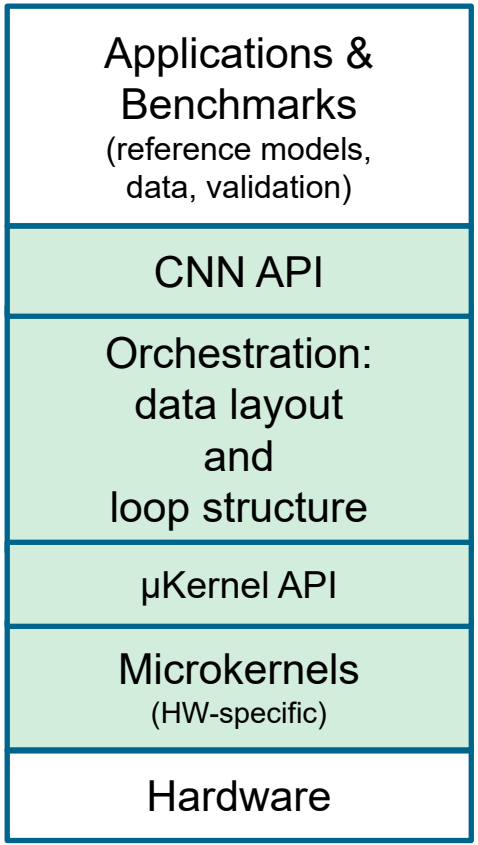
Upasana Sridhar
PhD, Electrical and Computer Engineering, CMU

Nicolai Tukanov
PhD, Electrical and Computer Engineering, CMU

Pankti Rajesh Shah
ECE master's student

Navya Chandra
ECE master's independent study: "Fused convolution on Pi Pico"

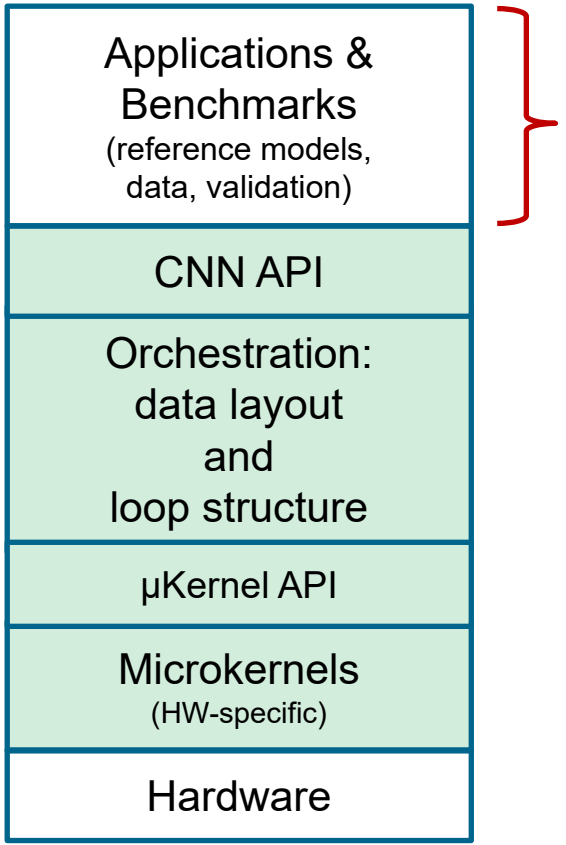
SMaLL: Software for Machine Learning Libraries



Approach: Two APIs:

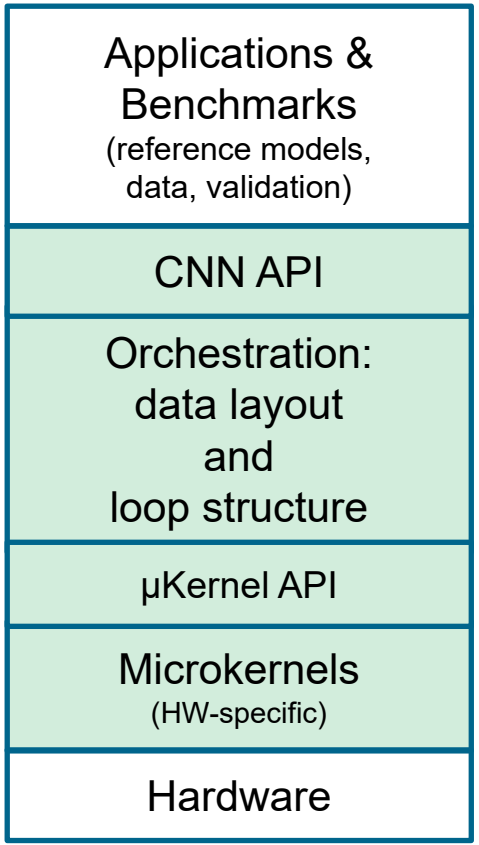
- **Usability**: A high-level CNN API provides common functionality for machine learning developers.
- **Performance-portability**: A low-level micro-kernel API defines a small number of primitives to be hand-optimized by hardware experts for specific hardware.

SMaLL: Software for Machine Learning Libraries



Approach: Prioritizing support for object detection and image classification models

SMaLL: Software for Machine Learning Libraries



Neural network layers currently implemented in the high-level SMaLL Library API:

- Convolution, partial and group
- 1x1 Convolution
- Depth-wise Convolution
- Max Pooling
- Activation (ReLU)
- Fully Connected (FC)—implemented as GEMM (or MMM) or 1x1 direct convolution

Coverage of MLPerf 'Tiny' and 'Mobile' Benchmarks

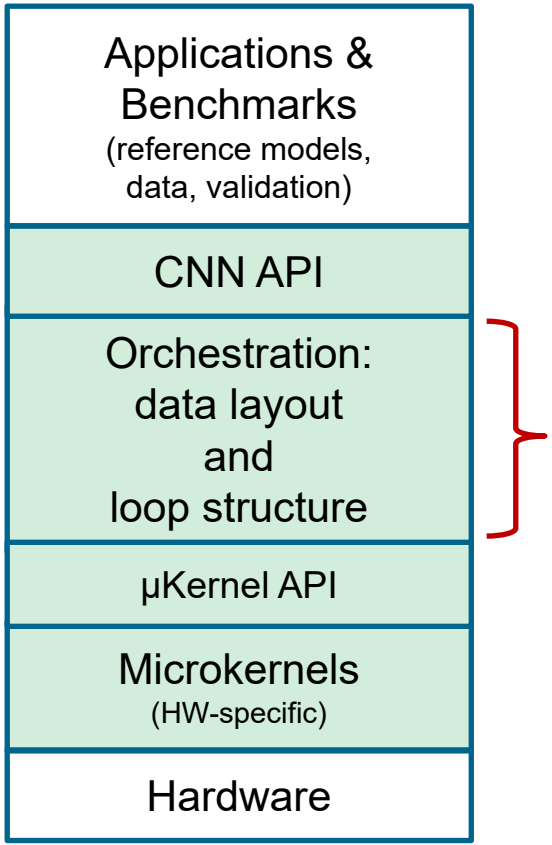
Approach: Prioritizing the object detection and image classification models

- Yellow → some layers not yet supported (e.g., Upsampling Convolution)
- Red → model type requires more study (e.g., Embedding Layers, Attention)

Task	Models	Dataset
Keyword Spotting	DS-CNN	Speech Commands
Visual Wake Words	MobileNet	Visual Wake Words Dataset
Image Classification	ResNet	Cifar10
Anomaly Detection	Deep AutoEncoder	ToyADMOS
Image classification	MobileNetEdgeTPU	ImageNet
Object detection	MobileDETs	MS-COCO 2017
Segmentation	DeepLabV3+	ADE20K (32 classes, 512x512)
Segmentation	MOSAIC (U-Net)	ADE20K (32 classes, 512x512)
Language processing	Mobile-BERT	SQUAD 1.1

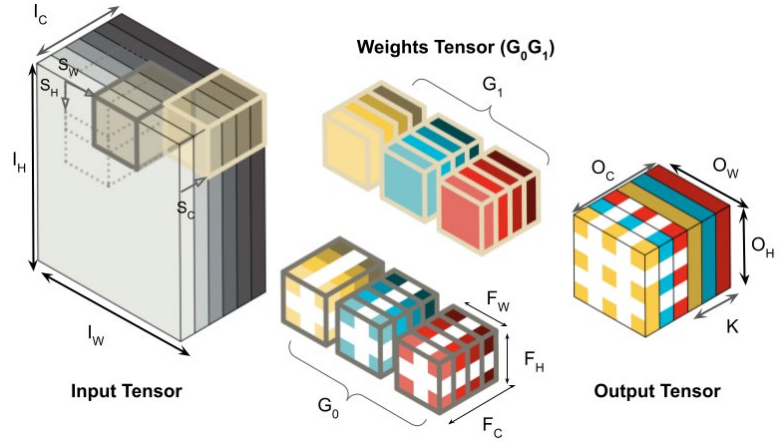
https://mlcommons.org

SMaLL: Software for Machine Learning Libraries

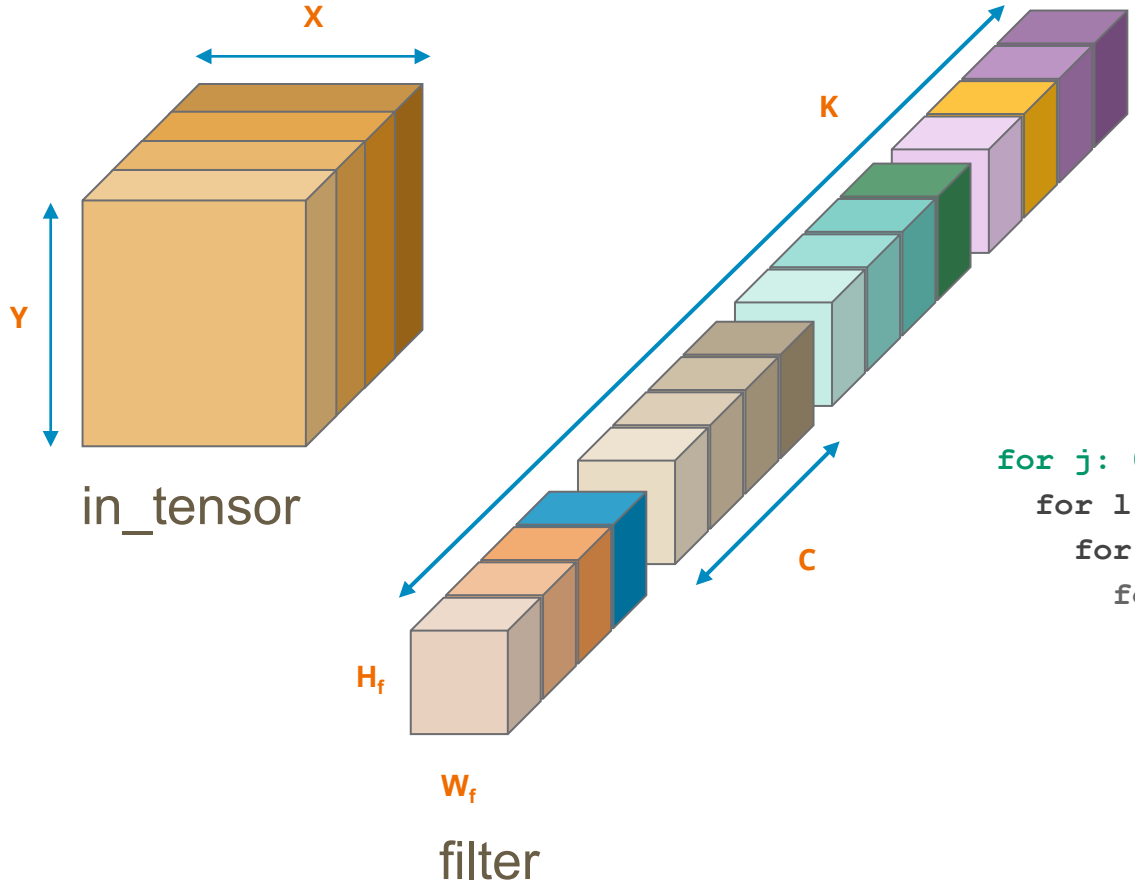


Approach: applying recent research advances in optimized computation

- "managing" the data
- "orchestrating" the computation



Convolution Operation—End to End




```
for j: 0 to K
  for l: 0 to Y
    for k: 0 to X
      for i: 0 to C
        for n: 0 to Hf
          for m: 0 to Wf
            output_tensor[j][l][k] +=
                (in_tensor[i][l * s + n][k * s + m]
                 * filter[j][i][n][m])
```

Orchestration: Data Layout and Loop Structure

High Performance Zero-Memory Overhead Direct Convolutions

Jiyuan Zhang [1] Franz Franchetti [1] Tze Meng Low [1]

Abstract

The computation of convolution layers in deep neural networks typically rely on high performance routines that trade space for time by using additional memory (either for packing purposes or required as part of the algorithm) to improve performance. The problems with such an approach are two-fold. First, these routines incur additional memory overhead which reduces the overall size of the network that can fit on embedded devices with limited memory capacity. Second, these high performance routines were not optimized for performing convolution, which means that the performance obtained is usually less than conventionally expected. In this paper, we demonstrate that direct convolution, when implemented *correctly*, eliminates all memory overhead, and yields performance that is between 10% to 400% times

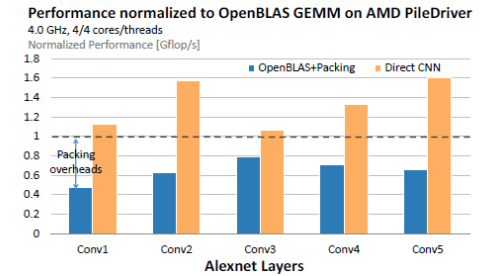


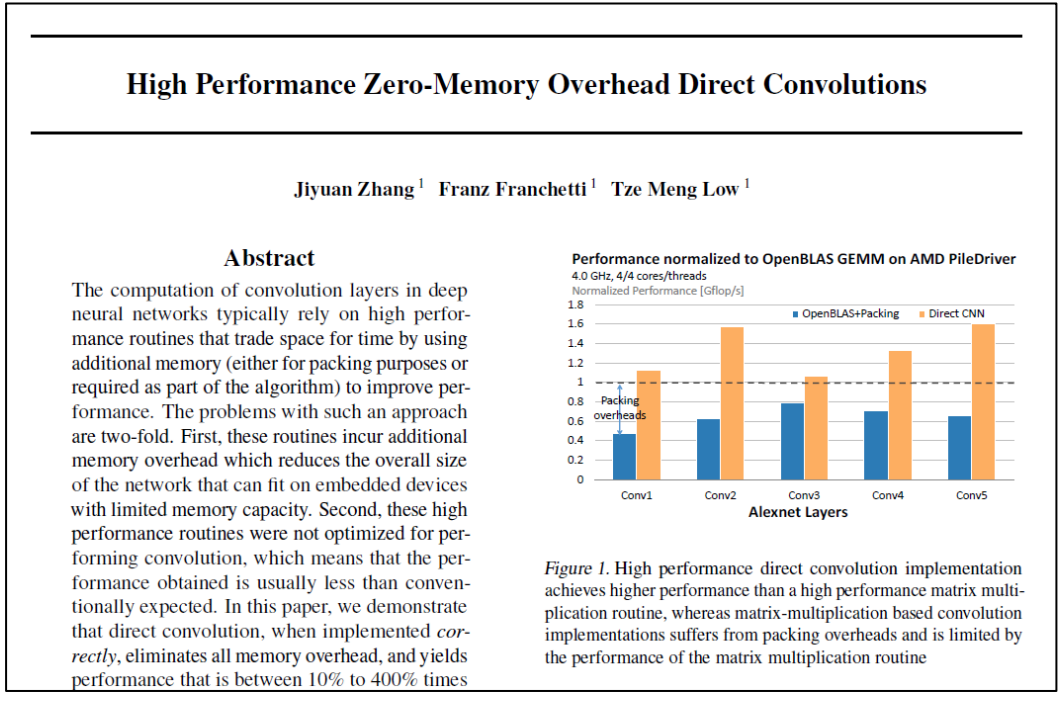
Figure 1. High performance direct convolution implementation achieves higher performance than a high performance matrix multiplication routine, whereas matrix-multiplication based convolution implementations suffers from packing overheads and is limited by the performance of the matrix multiplication routine

In *International Conference on Machine Learning*, pp. 5776-5785. PMLR, 2018.

Much of our efforts are targeted at extending CMU's 2018 research on direct convolutions.

- Custom data layout instead of packing
  - Saves memory
  - Blocks data for memory hierarchy
- Direct convolution loop nest is more computationally efficient

Orchestration: Data Layout and Loop Structure



High Performance Zero-Memory Overhead Direct Convolutions

Jiyuan Zhang [1] Franz Franchetti [1] Tze Meng Low [1]

Abstract

The computation of convolution layers in deep neural networks typically rely on high performance routines that trade space for time by using additional memory (either for packing purposes or required as part of the algorithm) to improve performance. The problems with such an approach are two-fold. First, these routines incur additional memory overhead which reduces the overall size of the network that can fit on embedded devices with limited memory capacity. Second, these high performance routines were not optimized for performing convolution, which means that the performance obtained is usually less than conventionally expected. In this paper, we demonstrate that direct convolution, when implemented *correctly*, eliminates all memory overhead, and yields performance that is between 10% to 400% times

Figure 1. High performance direct convolution implementation achieves higher performance than a high performance matrix multiplication routine, whereas matrix-multiplication based convolution implementations suffers from packing overheads and is limited by the performance of the matrix multiplication routine

In *International Conference on Machine Learning*, pp. 5776-5785. PMLR, 2018.

```
for j': 0 to Co/Cob in parallel
  for i': 0 to Ci/Cib
    for l: 0 to Ho
      for k': 0 to Wo/Wob
        for n: 0 to Hf
          for m: 0 to Wf
            for ii: 0 to Cib
              for kk: 0 to Wob
                for jj: 0 to Cob
                  out_tensor[j'*Cob+jj]
                            [k'*Wob+kk][l] +=
                    (in_tensor[i'*Cib + ii]
                              [s*k'*Wob + kk + m]
                              [l*s + n] *
                     filter[i'*Cib + ii]
                           [j'*Cob + jj][m][n])
```

Now 9 loops; outer loop is parallelized

Orchestration: Data Layout and Loop Structure

High Performance Zero-Memory Overhead Direct Convolutions

Jiyuan Zhang [1] Franz Franchetti [1] Tze Meng Low [1]

Abstract

The computation of convolution layers in deep neural networks typically rely on high performance routines that trade space for time by using additional memory (either for packing purposes or required as part of the algorithm) to improve performance. The problems with such an approach are two-fold. First, these routines incur additional memory overhead which reduces the overall size of the network that can fit on embedded devices with limited memory capacity. Second, these high performance routines were not optimized for performing convolution, which means that the performance obtained is usually less than conventionally expected. In this paper, we demonstrate that direct convolution, when implemented *correctly*, eliminates all memory overhead, and yields performance that is between 10% to 400% times

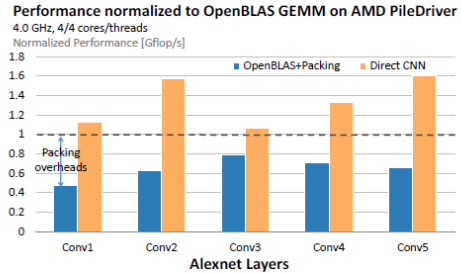


Figure 1. High performance direct convolution implementation achieves higher performance than a high performance matrix multiplication routine, whereas matrix-multiplication based convolution implementations suffers from packing overheads and is limited by the performance of the matrix multiplication routine

In *International Conference on Machine Learning*, pp. 5776-5785. PMLR, 2018.

```
for j': 0 to Co/Cob in parallel
  for i': 0 to Ci/Cib
    for l: 0 to Ho
      for k': 0 to Wo/Wob
        for n: 0 to Hf
          for m: 0 to Wf
            for ii: 0 to Cib
              for kk: 0 to Wob
                for jj: 0 to Cob
                  out_tensor[j'*Cob+jj]
                            [k'*Wob+kk][l] +=
                    (in_tensor[i'*Cib + ii]
                              [s*k'*Wob + kk + m]
                              [l*s + n] *
                     filter[i'*Cib + ii]
                           [j'*Cob + jj][m][n])
```

Three tuning parameters to block data for different hardware platforms

Orchestration: Fusing Convolution Layers

Beyond Element-wise Fusion for Reducing Convolutional Neural Nets Sizes

Upasana Sridhar*, Navya Chandra*, Martin Schatz+, Scott McMillan^, Tze Meng Low*
+ Meta, Inc. (Facebook)
^ Software Engineering Institute, Carnegie Mellon University
* Dept. of Electrical and Computer Engineering, Carnegie Mellon University

ABSTRACT

Fusing multiple layers in a deep learning network is commonly recognized as an approach to improve performance and reduce the amount of memory required. However, current approaches to fused layers are often limited to those that contain element-wise operations, such as Activation and Batch Normalization. More complicated layers are often not fused as the indexing overheads are often considered to be more costly than the benefits of fused layers. In this work, we show that fusing non-elementwise operations can be beneficial. Fundamental to our approach is the ability to express CNN layers using the same loop nest; simplifying the analysis and thus making it easier to specify how to fuse layers together. We show that this fusion produces a 1.5-10x reduction in the memory requirement. Moreover, we show that the fused implementations also produce a runtime improvement on the order of 4.6x - 9.9x compared to PyTorch and 1.2x - 20.2x compared to Tensorflow compiled with XLA.

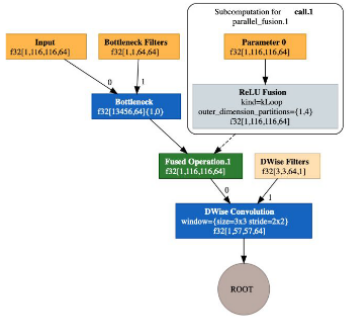


Figure 1: Post-fusion compute graph for Bottleneck (1 × 1) Convolution + ReLu Activation + Depthwise Convolution block using XLA demonstrating that fusion in XLA is limited. The Relu layer has been fused with the Bottleneck convo-

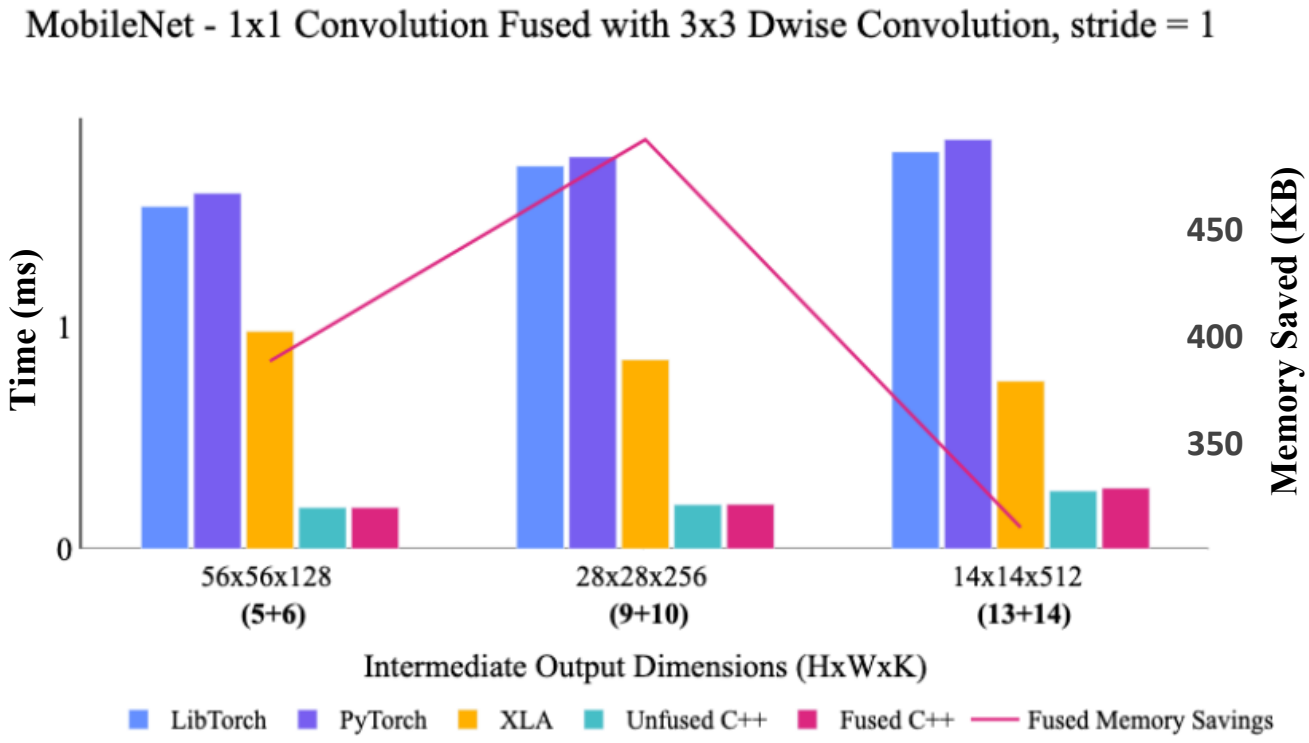
Algorithm 2 Fused abstract deep learning layer loop structure

```
1:  for g ≤ G do
2:      for j ≤ K do
3:
4:          for i ≤ F_C do                          ▷ Fusion in Last iteration
5:
6:              for k ≤ O_h do    ▷ Transformation Req for Fusion
7:                  for l ≤ O_w do
8:                      for x ≤ F_h do
9:                          for y ≤ F_w do
10:                             for ii ≤ I_cb do
11:
12:                                 for ll ≤ O_wb do
13:                                     for jj ≤ O_cb do
14:                                         ReductionOp(O_0, I, F_0)
15:                                     end for
16:                                 end for
17:                                     ▷ Single Element Reduction
18:                                 for ll ≤ O_wb do
19:                                     for jj ≤ O_cb do
20:                                         ReductionOp(O, O_0, F_1)
21:                                     end for
22:                                 end for
23:                             end for
24:                         end for
25:                     end for
26:                                         ▷ Channel Reduction
27:                     for x ≤ F_h^1 do
28:                         for y ≤ F_w^1 do
29:                             ii = 1
30:                             for ll ≤ O_wb do
31:                                 for jj ≤ O_cb do
32:                                     ReductionOp(O, O_0, F_1)
33:                                 end for
34:                             end for
35:                         end for
36:                     end for
37:                 end for
38:             end for
39:         end for
40:     end for
41: end for=0
```

Orchestration: Fusing Convolution Layers

New research results on combining (or fusing) neural network layers.

- Specifically targeting convolution layers
- **1.5x–10x** memory reduction
- **1.2x – 20x** performance gains over pytorch and Tensorflow



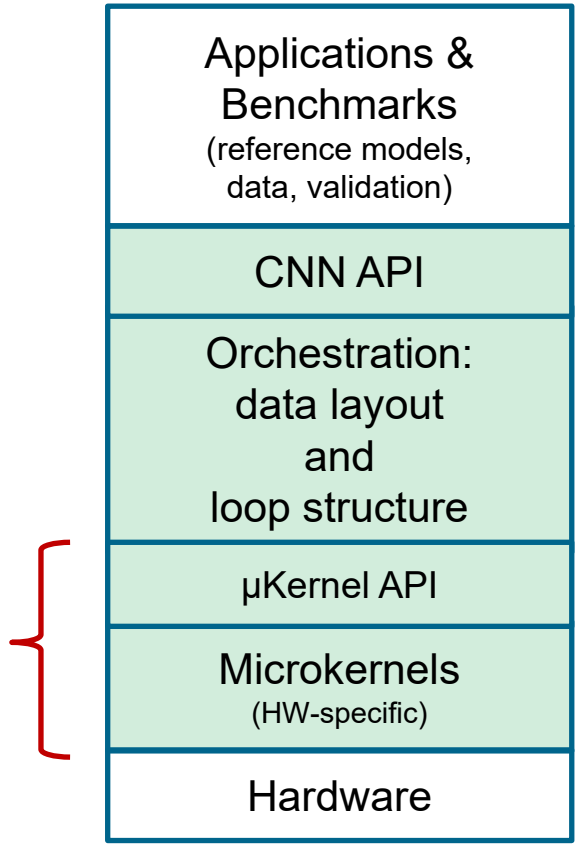
SMaLL: Software for Machine Learning Libraries

Innermost loops define the microkernels the low-level API.

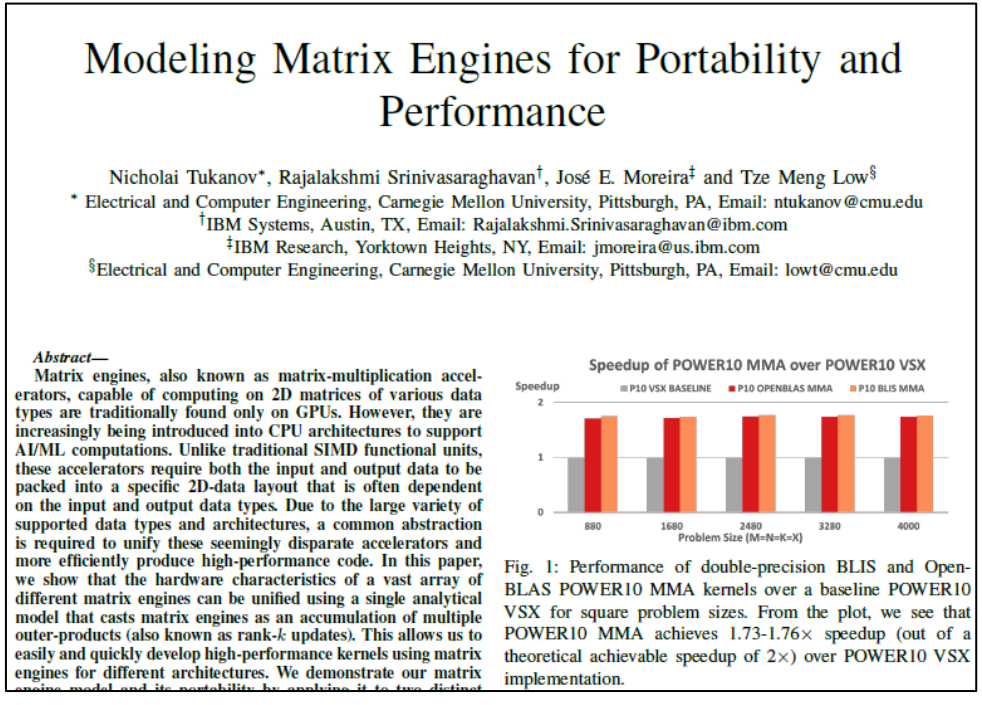
Microkernels are developed for specific targeted hardware (sometimes in assembly code).

Performance models developed from experiments using microkernels.

These models inform the selection of the **data blocking factors** in the orchestration layer.



Publications



Modeling Matrix Engines for Portability and Performance

Nicholai Tukanov*, Rajalakshmi Srinivasaraghavan†, José E. Moreira‡ and Tze Meng Low§
* Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, Email: ntukanov@cmu.edu
†IBM Systems, Austin, TX, Email: Rajalakshmi.Srinivasaraghavan@ibm.com
‡IBM Research, Yorktown Heights, NY, Email: jmoreira@us.ibm.com
§Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, Email: lowt@cmu.edu

***Abstract*—**
Matrix engines, also known as matrix-multiplication accelerators, capable of computing on 2D matrices of various data types are traditionally found only on GPUs. However, they are increasingly being introduced into CPU architectures to support AI/ML computations. Unlike traditional SIMD functional units, these accelerators require both the input and output data to be packed into a specific 2D-data layout that is often dependent on the input and output data types. Due to the large variety of supported data types and architectures, a common abstraction is required to unify these seemingly disparate accelerators and more efficiently produce high-performance code. In this paper, we show that the hardware characteristics of a vast array of different matrix engines can be unified using a single analytical model that casts matrix engines as an accumulation of multiple outer-products (also known as rank-k updates). This allows us to easily and quickly develop high-performance kernels using matrix engines for different architectures. We demonstrate our matrix

Fig. 1: Performance of double-precision BLIS and OpenBLAS POWER10 MMA kernels over a baseline POWER10 VSX for square problem sizes. From the plot, we see that POWER10 MMA achieves 1.73-1.76× speedup (out of a theoretical achievable speedup of 2×) over POWER10 VSX implementation.

IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2022

[Published] "Modeling Matrix Engines for Portability and Performance." *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. May 2022.

[Submitted] "SMaLL: Software for Rapidly Developing Machine Learning Libraries." *ACM Transactions on Embedded Computing: Special issue on TinyML*.

Carnegie Mellon University
Software Engineering Institute

What's Next

- Developing microkernels for ATLAS hardware platform and benchmarking
- Adding support for more neural network layers
- Implementing all possible fused layers
- Open-source software release
- Explore integration with the MLIR ecosystem

For further information:
Scott McMillan
info@sei.cmu.edu