tinyML® Summit

Enabling Ultra-low Power Machine Learning at the Edge

February 12-13, 2020
Burlingame, California

www.tinyML.org
Using ML for ML to span the gamut of TinyML hardware

TinyML 2020

Jason Knight - Co-founder and CPO at OctoML
## Motivation

- Cambrian explosion of models, workloads, and use cases.
- Rapidly evolving ML software ecosystem
- Silicon scaling limitations (Dennard and Moore):
  - Cambrian explosion of HW backends.
  - Heterogeneous HW.
- Growing set of requirements:
  - Cost
  - Latency
  - Power
  - Security
Motivation

Cambrian explosion of models, workloads, and use cases.

Rapidly evolving ML software ecosystem

Silicon scaling limitations (Dennard and Moore):
Cambrian explosion of HW backends.
Heterogeneous HW.

Growing set of requirements:

Hard enough on server class hardware...

But now we want to do it in on **bare-metal** devices?
Bare-Metal Devices

FPGA

ARM M class,
RISC-V MCUs, etc
Programming bare-metal devices is cumbersome
Running ML on bare-metal hardware is hard
Running ML on bare-metal hardware is hard

ML Model → GCC → ML from scratch in C... → Bare-metal Hardware
Running ML on bare-metal hardware is pretty hard
Running ML on bare-metal hardware is pretty hard

Slow kernels, Incomplete coverage

ML Model ➔ ML Framework ➔ GCC ➔ Hardware
Running ML on bare-metal hardware is **really hard**.
Running ML on bare-metal hardware is **really hard**

ML Model → ML Framework → Hand-optimize kernels → GCC

Not much help here…
Running ML on bare-metal hardware is *really hard*
Running ML on bare-metal hardware is really hard
Optimizing ML on bare-metal hardware is a long road
μTVM simplifies ML software on bare-metal
Our Goal

Minimize the effort to run and optimize ML on bare-metal.
Our Goal

Minimize the effort to run and optimize ML on bare-metal.

So what’s our approach?
The TVM in μTVM
The TVM in µTVM

![Icons](image)
The TVM in μTVM

How do we get from here…

to here?

FPGA  ASIC
The TVM in µTVM
The TVM in µTVM
The TVM in µTVM

High-Level Differentiable IR

FPGA  ASIC
The TVM in μTVM

High-Level Differentiable IR

Tensor Expression IR

???

FPGA  ASIC
The TVM in μTVM

High-Level Differentiable IR

Tensor Expression IR

LLVM, CUDA

VTA

FPGA

ASIC
The TVM in μTVM

- High-Level Differentiable IR
- Tensor Expression IR
- LLVM, CUDA
- VTA
- FPGA
- ASIC

Hand-optimization?
The TVM in µTVM
Where TVM is being used today

Every “Alexa” wake-up today across all devices uses a model optimized with TVM

“[TVM enabled] real-time on mobile CPUs for free...We are excited about the performance TVM achieves.” More than 85x speed-up for speech recognition model.

Bing query understanding: 112ms (Tensorflow) -> 34ms (TVM). QnA bot: 73ms->28ms (CPU), 10.1ms->5.5ms (GPU)

“TVM is key to ML Access on Hexagon”
Where TVM is being used today

Every “Alexa” wake-up today across all devices uses a model optimized with TVM

“[TVM enabled] real-time on mobile CPUs for free...We are excited about the performance TVM achieves.” More than 85x speed-up for speech recognition model.

Bing query understanding: 112ms (Tensorflow) -> 34ms (TVM). QnA bot: 73ms->28ms (CPU), 10.1ms->5.5ms (GPU)

“TVM is key to ML Access on Hexagon”

Cf: TVM Conference 2019 (Dec) - videos and slides available online
The TVM in µTVM

Many hardware targets already enjoy speedups from TVM
Motivation

Except for microcontrollers...

What about us? 😞
Today’s Talk

High-Level Differentiable IR

Tensor Expression IR

μTVM

LLVM, CUDA

VTA

FPGA

ASIC

AutoTVM
Today’s Talk

High-Level Differentiable IR

Tensor Expression IR

μTVM

LLVM, CUDA

VTA

FPGA

ASIC

AutoTVM
Today’s Talk

High-Level Differentiable IR

Tensor Expression IR

μTVM

LLVM, CUDA

VTA

FPGA

ASIC

AutoTVM
Today’s Talk

High-Level Differentiable IR

Tensor Expression IR

μTVM

LLVM, CUDA

VTA

FPGA

ASIC

AutoTVM
AutoTVM

TVM + Program
AutoTVM

TVM + Program → compile program → Hardware Backend

run and measure
AutoTVM

TVM + Program  \(\xrightarrow{\text{compile program}}\)  Hardware Backend

AutoTVM  \(\xleftarrow{\text{return timing feedback}}\)  Hardware Backend

run and measure
AutoTVM

- TVM + Program → Hardware Backend
  - compile program
  - run and measure
  - return timing feedback
  - improve implementation

AutoTVM
AutoTVM

1. TVM + Program
   - compile program
   - improve implementation

2. Hardware Backend
   - run and measure
   - return timing feedback

3. AutoTVM
AutoTVM

TVM + Program → compile program → Hardware Backend

improve implementation

return timing feedback

run and measure
AutoTVM

TVM + Program → Hardware Backend
compile program

AutoTVM
improve implementation

return timing feedback

run and measure
AutoTVM

Automatically adapt to hardware type by learning

![Diagram showing AutoTVM process]

- **Expression (e)**
  - **Search Space (S_e)**
  - **AutoTVM**
    - **Statistical Cost Model**
    - **Code Generator**
  - **Code (c)**
  - **Execution (x)**
  - **Training data (D)**
  - **Function f(x)**
Today’s Talk

High-Level Differentiable IR

Tensor Expression IR

μTVM

LLVM, CUDA

VTA

μTVM

FPGA

ASIC

AutoTVM

1

2

3
Today’s Talk

μTVM

Using μTVM
μTVM Runtime
Graph Runtime
Today’s Talk

µTVM

Using µTVM

µTVM Runtime

Graph Runtime
How easy is it to use μTVM?
How easy is it to use μTVM?

- Implement an interface to:
  - Read and write to device memory
  - Start code execution
How easy is it to use μTVM?

• Implement an interface to:
  - Read and write to device memory
  - Start code execution

• Provide a cross-compiler for the device
How easy is it to use μTVM?

• Implement an interface to:
  - Read and write to device memory
  - Start code execution

• Provide a cross-compiler for the device
ResNet Example in Mainline TVM
ResNet Example in Mainline TVM

image =
ResNet Example in Mainline TVM

```python
image = ...
resnet, params = get_resnet()
module = graph_runtime_build(resnet, params)
```
ResNet Example in Mainline TVM

```python
image = 'cat'
resnet, params = get_resnet()
module = graph_runtime_build(resnet, params)
module.run(image)
```

> 'cat'
TVM $\rightarrow$ μTVM

```python
image = resnet, params = get_resnet()
module = graph_runtime_build(resnet, params)
```

```python
image = resnet, params = get_resnet()
with micro.Session('openocd') as sess:
    module = sess.build(resnet, params)
```
TVM $\rightarrow$ $\mu$TVM

```python
image = resnet, params = get_resnet()
module = graph_runtime_build(resnet, params)
```

wrap in a session

```python
image = resnet, params = get_resnet()
with micro.Session('openocd') as sess:
    module = sess.build(resnet, params)
```
TVM $\rightarrow$ µTVM

```python
image = resnet, params = get_resnet()
module = graph_runtime_build(resnet, params)
```

wrap in a session

```python
image = resnet, params = get_resnet()
with micro.Session('openocd') as sess:
    module = sess.build(resnet, params)
```

different build function
Today’s Talk

µTVM

Using µTVM  µTVM Runtime  Graph Runtime
Today’s Talk

μTVM

Using μTVM
μTVM Runtime
Graph Runtime

µTVM
Today’s Talk

μTVM Runtime

C Code Generator

μDevice Interface
Code Generation

High-Level Differentiable IR

Tensor Expression IR

μTVM

LLVM, CUDA

VTA

μTVM

FPGA

ASIC

AutoTVM

CUDA

FPGA

ASIC
Code Generation

High-Level Differentiable IR

Tensor Expression IR

why not LLVM?

μTVM

LLVM, CUDA

VTA

FPGA

ASIC

AutoTVM
Code Generation

High-Level Differentiable IR

Tensor Expression IR

μTVM

LLVM, CUDA

VTA

FPGA, ASIC

AutoTVM

lack of support...
Code Generation

High-Level Differentiable IR

Tensor Expression IR

AutoTVM

C
LLVM, CUDA
VTA
µTVM
FPGA
ASIC

OctoML
µDevice Interface

Read

Write

Execute
µDevice Interface


Read  Write  Execute
The Runtime

μTVM Runtime

C Code Generator

μDevice Interface
The Runtime in Action

μTVM Runtime

C Code Generator

μDevice Interface
Set up a RWX interface with the board
Code generator lowers TVM IR to C code

μTVM Runtime

C Code Generator  μDevice Interface

communication interface

IR → code

func.c
Device compiler cross compiles generated code

μTVM Runtime

| C Code Generator | μDevice Interface |

communication interface

IR → code

gcc

func.c → func.o
μTVM remaps generated binary to custom device memory locations for multiple library loading.
Remapped binary is loaded onto device

μTVM Runtime

C Code Generator  μDevice Interface

communication interface

custom loader

IR → code

gcc

func.c

func.o

1d linker

remap func

remap func

OctoML
Function invocation on host transfers parameters to the device for execution

- **μTVM Runtime**
  - C Code Generator
  - μDevice Interface

- **Send parameters**

- **IR → code**
  - `func.c` → `func.o`

- **Custom loader**
  - `gcc` → `func.o`
  - `ld` → `remap func`

- **Run**
  - "'cat'"
Today’s Talk

High-Level Differentiable IR

Tensor Expression IR

μTVM

LLVM, CUDA

VTA

FPGA

ASIC

AutoTVM
AutoTVM on µTVM

- Same pipeline as usual
- Load kernels into RAM instead of flash
End-to-End CIFAR-10 Evaluation

Replicated an int8-quantized CNN from an ARM Mbed tutorial
Preliminary CIFAR-10 CNN Results

- Ran on ARM Cortex-M7
- Compared against CMSIS-NN
- Vanilla template
- ~5 hours of tuning
- **No vectorization**

![Graph showing comparison between MicroTVM Untuned, MicroTVM Tuned, and CMSIS-NN. MicroTVM Untuned has a time of 612 ms, MicroTVM Tuned has a time of 241 ms, and CMSIS-NN has a time of 126 ms.]
Preliminary Int-8 Conv2D Results

Fast Int-8 Conv2D

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroTVM Untuned</td>
<td>321</td>
</tr>
<tr>
<td>MicroTVM Tuned</td>
<td>122</td>
</tr>
<tr>
<td>MicroTVM SIMD Tuned</td>
<td>84</td>
</tr>
<tr>
<td>CMSIS-NN</td>
<td>50</td>
</tr>
</tbody>
</table>

RGB Int-8 Conv2D

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroTVM Untuned</td>
<td>125</td>
</tr>
<tr>
<td>MicroTVM Tuned</td>
<td>55</td>
</tr>
<tr>
<td>CMSIS-NN</td>
<td>35</td>
</tr>
</tbody>
</table>
Coming Soon - Self-hosted models
Coming Soon - Ultra low bit-width quantization

- TVM already supports flexible code generation for a variety of data types
- Natural to combine this with µTVM
- See Josh Fromm’s MLSys talk on March 3rd to learn more
Coming soon - µTVM as a service

- OctoML offering hosted version of AutoTVM for cloud, mobile, and embedded
  - Cloud CPU/GPU
  - ARM A class CPU/GPU
  - ARM M class microcontrollers
  - More on request
- Trained model in, optimized binary out
- Currently in private beta

Octomizer

TensorFlow, Pytorch, ONNX serialized models

Optimized deployment artifacts

API and web UI

Auto-tuning using OctoML clusters
Status and next steps

- Functional on x86, ARM, and RISC-V
- In-depth writeup early March on our blog with upstream TVM documentation/tutorial
  - Follow us on https://octoml.ai or @octoml
- Interop with CMSIS-NN library and TF Lite Micro (better together!)
- First class SIMD vectorization support for ARM schedules, (in addition to current tensorization approach)
Questions or interest?
Please reach out:

Jason Knight
jknights@octoml.ai

Acknowledgments

Logan Weber - AutoTVM
Joshua Fromm - RipTide
## Accuracy / Runtime

<table>
<thead>
<tr>
<th>Model</th>
<th>Name</th>
<th>1-bit</th>
<th>2-bit</th>
<th>3-bit</th>
<th>full precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ImageNet top-1 accuracy / Runtime (ms)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>AlexNet Xnor-Net [48]</td>
<td>44.2%</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>AlexNet BNN [12]</td>
<td>27.9%</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>AlexNet DoReFaNet [63]</td>
<td>43.6%</td>
<td>49.8%</td>
<td>48.4%</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>AlexNet QNN [27]</td>
<td>43.3%</td>
<td>51.0%</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>AlexNet HWGQ [4]</td>
<td>—</td>
<td>52.7%</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>6</td>
<td>VGGNet HWGQ [4]</td>
<td>—</td>
<td>64.1%</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>AlexNet Riptide-unipolar (ours)</td>
<td>44.5%</td>
<td>52.5%</td>
<td>53.6%</td>
<td>282.8</td>
</tr>
<tr>
<td>8</td>
<td>AlexNet Riptide-bipolar (ours)</td>
<td>42.8%</td>
<td>50.4%</td>
<td>52.4%</td>
<td>207.0</td>
</tr>
<tr>
<td>9</td>
<td>VGGNet Riptide-unipolar (ours)</td>
<td>56.8%</td>
<td>64.2%</td>
<td>67.1%</td>
<td>610.0</td>
</tr>
<tr>
<td>10</td>
<td>VGGNet Riptide-bipolar (ours)</td>
<td>54.4%</td>
<td>61.5%</td>
<td>65.2%</td>
<td>423.5</td>
</tr>
<tr>
<td>11</td>
<td>ResNet18 Riptide-unipolar (ours)</td>
<td>47.9%</td>
<td>58.4%</td>
<td>61.8%</td>
<td>152.3</td>
</tr>
</tbody>
</table>
Copyright Notice

The presentation(s) in this publication comprise the proceedings of tinyML® Summit 2020. The content reflects the opinion of the authors and their respective companies. This version of the presentation may differ from the version that was presented at the tinyML Summit. The inclusion of presentations in this publication does not constitute an endorsement by tinyML Foundation or the sponsors.

There is no copyright protection claimed by this publication. However, each presentation is the work of the authors and their respective companies and may contain copyrighted material. As such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

tinyML is a registered trademark of the tinyML Foundation.

www.tinyML.org