

CMSC 5743



Efficient Computing of Deep Neural Networks

Lecture 09: TVM

Bei Yu

CSE Department, CUHK

byu@cse.cuhk.edu.hk

(Latest update: October 4, 2021)

Fall 2021



These slides contain/adapt materials developed by

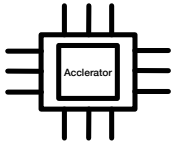
Chen, Tianqi, et al. "TVM: An automated end-to-end optimizing compiler for deep learning." 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018.



Beginning of Story

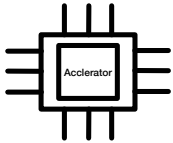


Beginning of Story



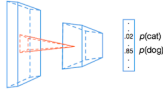


Beginning of Story





Beginning of Story



```
// Pseudo-code for convolution program for the VIA accelerator
// Virtual Thread 0
0x00: LOAD(PARAM[ 0-71]) // LDMT00
0x01: LOAD(ACTIV[ 0-24]) // LDMT00
0x02: LOAD(LDBUF[ 0-31]) // LDMT00
0x03: PUSH(LD->EX) // EXMT00
0x04: POP(LD->EX) // EXMT00
0x05: EXE (ACTIV[ 0-24],PARAM[ 0-71],LDBUF[ 0-31],STBUF[ 0-7]) // EXGT00
0x06: PUSH(EX->LD) // EXMT00
0x07: PUSH(EX->ST) // EXMT00
0x08: POP(EX->ST) // STMT00
0x09: STOR(STBUF[ 0-7]) // STMT00
0x0A: PUSH(ST->EX) // STMT00
// Virtual Thread 1
0x0B: LOAD(ACTIV[25-50]) // LDMT01
0x0C: LOAD(LDBUF[32-63]) // LDMT01
0x0D: PUSH(LD->EX) // LDMT01
0x0E: POP(LD->EX) // EXMT01
0x0F: EXE (ACTIV[25-50],PARAM[ 0-71],LDBUF[32-63],STBUF[32-39]) // EXGT01
0x10: PUSH(EX->LD) // EXMT01
0x11: PUSH(EX->ST) // EXMT01
0x12: POP(EX->ST) // STMT01
0x13: STOR(STBUF[32-39]) // STMT01
0x14: PUSH(ST->EX) // STMT01
// Virtual Thread 2
0x15: POP(EX->LD) // LDMT02
0x16: LOAD(PARAM[ 0-71]) // LDMT02
0x17: LOAD(ACTIV[ 0-24]) // LDMT02
0x18: LOAD(LDBUF[ 0-31]) // LDMT02
0x19: PUSH(LD->EX) // LDMT02
0x1A: POP(LD->EX) // EXMT02
0x1B: POP(ST->EX) // EXMT02
0x1C: EXE (ACTIV[ 0-24],PARAM[ 0-71],LDBUF[ 0-31],STBUF[ 0-7]) // EXGT02
0x1D: PUSH(EX->ST) // EXMT02
0x1E: POP(EX->ST) // STMT02
0x1F: STOR(STBUF[ 0-7]) // STMT02
// Virtual Thread 3
0x20: POP(EX->LD) // LDMT03
0x21: LOAD(ACTIV[25-50]) // LDMT03
0x22: LOAD(LDBUF[32-63]) // LDMT03
0x23: PUSH(LD->EX) // LDMT03
0x24: POP(LD->EX) // EXMT03
0x25: POP(ST->EX) // EXMT03
0x26: EXE (ACTIV[25-50],PARAM[ 0-71],LDBUF[32-63],STBUF[32-39]) // EXGT03
0x27: PUSH(EX->ST) // EXMT03
0x28: POP(EX->ST) // STMT03
0x29: STOR(STBUF[32-39]) // STMT03
```

(a) Blocked convolution program with multiple thread contexts

```
// Convolution access pattern dictated by micro-coded program.
// Each register index is derived as a 2-D affine function.
// e.g.  $idx_{xy} = a_x y + b_x + c_x r$ , where  $c_x$  is specified by
// micro op B fields.
for y in [B.1]
  for x in [B.2]
     $rf[idx_{xy}] += GEV(act[idx_{xy}], par[idx_{xy}])$ 
     $rf[idx_{xy}] += GEV(act[idx_{xy}], par[idx_{xy}])$ 
  -
   $rf[idx_{xy}] += GEV(act[idx_{xy}], par[idx_{xy}])$ 
```

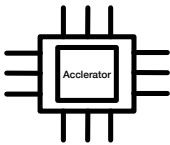


(b) Convolution micro-coded program

```
// Max-pool, batch normalization and activation function
// Access pattern dictated by micro-coded program.
// Each register index is derived as a 2D affine function.
// e.g.  $idx_{xy} = a_{xy} y + b_{xy} + c_{xy} r$ , where  $c_{xy}$  is specified by
// micro op B fields.
for x in [B.1]
  for y in [B.2]
    // max pooling
     $rf[idx_{xy}] = \text{MAX}(rf[idx_{xy}], rf[idx_{xy}])$ 
     $rf[idx_{xy}] = \text{MAX}(rf[idx_{xy}], rf[idx_{xy}])$ 
    // batch norm
     $rf[idx_{xy}] = \text{MUL}(rf[idx_{xy}], rf[idx_{xy}])$ 
     $rf[idx_{xy}+1] = \text{ADD}(rf[idx_{xy}], rf[idx_{xy}+1])$ 
     $rf[idx_{xy}+2] = \text{MUL}(rf[idx_{xy}+2], rf[idx_{xy}+2])$ 
     $rf[idx_{xy}+3] = \text{ADD}(rf[idx_{xy}+3], rf[idx_{xy}+3])$ 
  -
  // activation
   $rf[idx_{xy}+3] = \text{RELU}(rf[idx_{xy}+3], rf[idx_{xy}+3])$ 
   $rf[idx_{xy}+3] = \text{RELU}(rf[idx_{xy}+3], rf[idx_{xy}+3])$ 
```



(c) Max pool, batch norm and activation micro-coded program





Goal: Deploy Deep Learning Everywhere

Frameworks





Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks



Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Explosion of hardware backends



Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Explosion of hardware backends





Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Explosion of hardware backends





Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Explosion of hardware backends





Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Explosion of hardware backends





Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Explosion of hardware backends



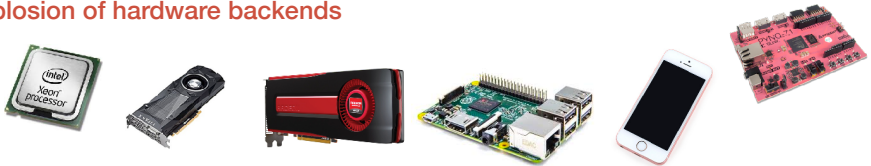


Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Explosion of hardware backends



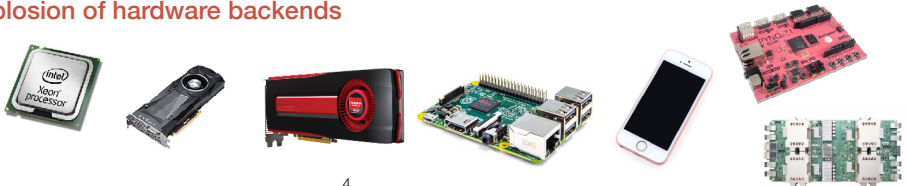


Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Explosion of hardware backends



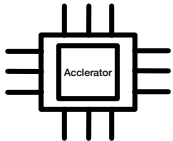


Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Explosion of hardware backends





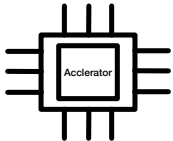
Goal: Deploy Deep Learning Everywhere



Explosion of models and frameworks

Huge gap between model/frameworks and hardware backends

Explosion of hardware backends





Existing Approach

Frameworks

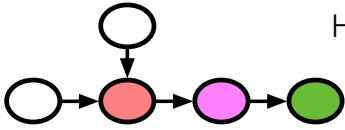


Hardware





Existing Approach



High-level data flow graph



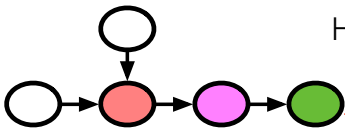


Existing Approach

Frameworks



High-level data flow graph



Primitive Tensor operators such as Conv2D

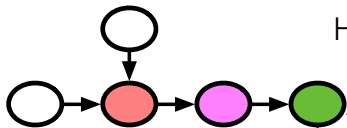
Hardware





Existing Approach

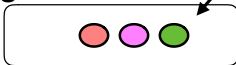
Frameworks



High-level data flow graph

Primitive Tensor operators such as Conv2D

eg. cuDNN



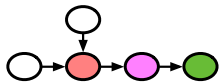
Offload to heavily optimized
DNN operator library

Hardware



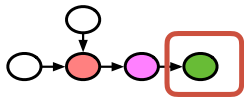


Existing Approach: Engineer Optimized Tensor Operators



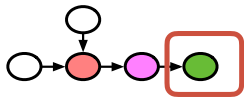


Existing Approach: Engineer Optimized Tensor Operators





Existing Approach: Engineer Optimized Tensor Operators

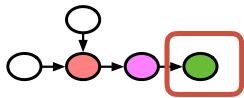


Matmul: Operator Specification

```
C = tvm.compute((m, n),  
                lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```



Existing Approach: Engineer Optimized Tensor Operators



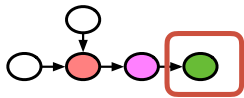
Matmul: Operator Specification

```
C = tvn.compute((m, n),  
    lambda y, x: tvn.sum(A[k, y] * B[k, x], axis=k))
```





Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

```
C = tvm.compute((m, n),  
                lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

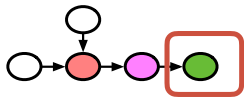


Vanilla Code

```
for y in range(1024):  
    for x in range(1024):  
        C[y][x] = 0  
        for k in range(1024):  
            C[y][x] += A[k][y] * B[k][x]
```



Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

```
C = tvm.compute((m, n),  
                lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

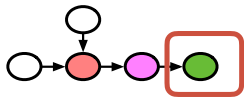


Loop Tiling for Locality

```
for yo in range(128):  
    for xo in range(128):  
        C[yo*8:yo*8+8][xo*8:xo*8+8] = 0  
        for ko in range(128):  
            for yi in range(8):  
                for xi in range(8):  
                    for ki in range(8):  
                        C[yo*8+yi][xo*8+xi] +=  
                            A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```




Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

```
C = tvm.compute((m, n),  
                lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```



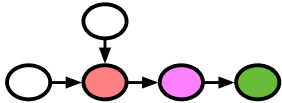
Map to Accelerators

```
inp_buffer AL[8][8], BL[8][8]  
acc_buffer CL[8][8]  
for yo in range(128):  
    for xo in range(128):  
        vdl.a.fill_zero(CL)  
        for ko in range(128):  
            vdl.a.dma_copy2d(AL, A[ko*8:ko*8+8][yo*8:yo*8+8])  
            vdl.a.dma_copy2d(BL, B[ko*8:ko*8+8][xo*8:xo*8+8])  
            vdl.a.fused_gemm8x8_add(CL, AL, BL)  
            vdl.a.dma_copy2d(C[yo*8:yo*8+8,xo*8:xo*8+8], CL)
```

Human exploration of optimized code



Limitations of Existing Approach

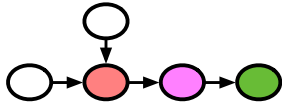


cuDNN





Limitations of Existing Approach

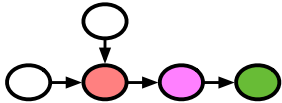


cuDNN

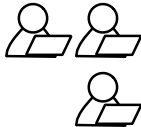




Limitations of Existing Approach

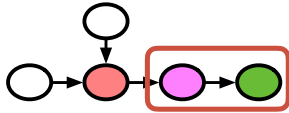


cuDNN

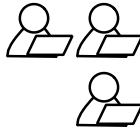




Limitations of Existing Approach

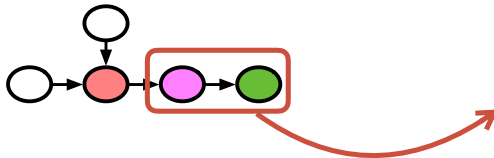


cuDNN

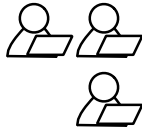




Limitations of Existing Approach

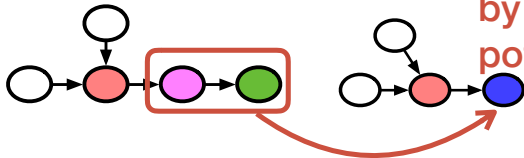


cuDNN





Limitations of Existing Approach



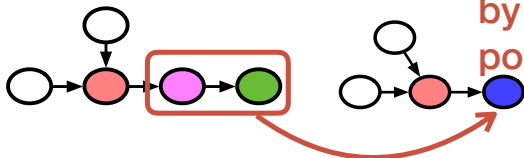
New operator introduced
by operator fusion optimization
potentially benefit: 1.5x speedup

cuDNN



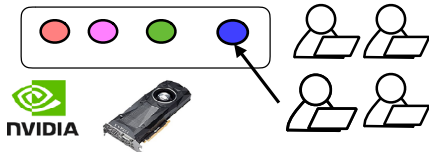


Limitations of Existing Approach



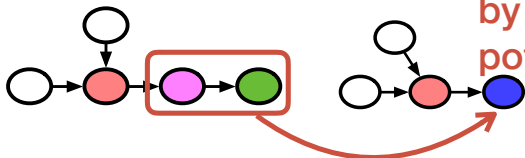
New operator introduced
by operator fusion optimization
potentially benefit: 1.5x speedup

cuDNN





Limitations of Existing Approach



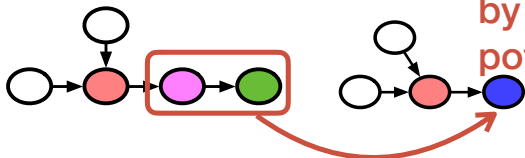
New operator introduced
by operator fusion optimization
potentially benefit: 1.5x speedup

cuDNN





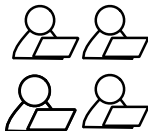
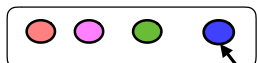
Limitations of Existing Approach



New operator introduced
by operator fusion optimization
potentially benefit: 1.5x speedup

Engineering intensive

cuDNN



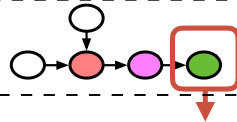
9





Learning-based Learning System

Frameworks



High-level data flow graph and optimizations



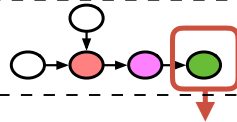
Hardware





Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

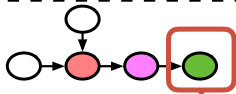
Hardware





Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

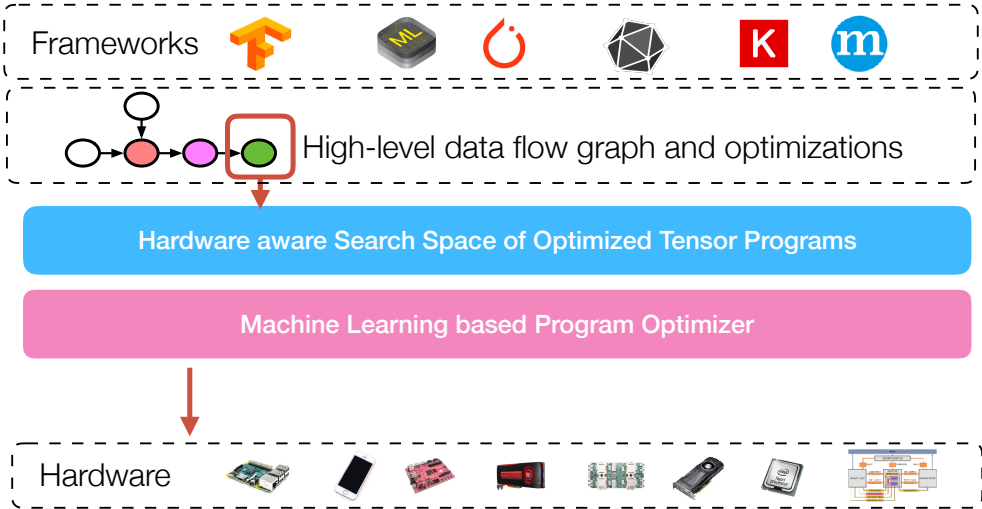
Hardware aware Search Space of Optimized Tensor Programs

Hardware





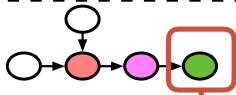
Learning-based Learning System





Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

directly generate optimized program
for new operator workloads and hardware

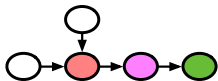
Hardware





Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

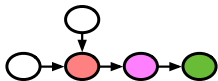
Hardware





Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

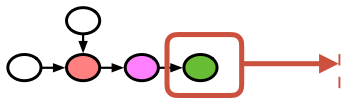
Machine Learning based Program Optimizer

Hardware





Hardware-aware Search Space



Tensor Expression Language (Specification)

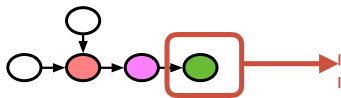
```
C = tvm.compute((m, n),  
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Hardware





Hardware-aware Search Space



Tensor Expression Language (Specification)

```
C = tvm.compute((m, n),  
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Define search space of hardware aware mappings from expression to hardware program

Based on Halide's compute/schedule separation

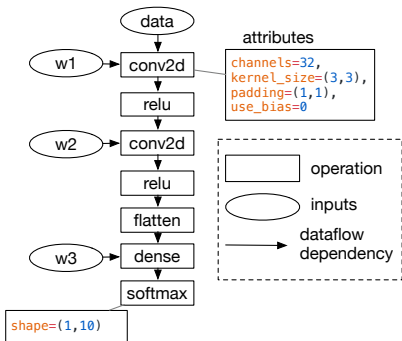
Hardware



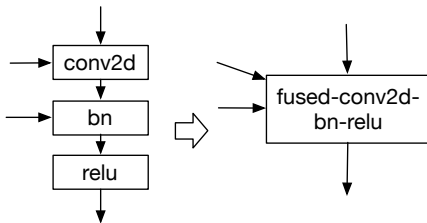


Computational Graph as IR

Represent High level
Deep Learning Computations



Effective Equivalent Transformations
to Optimize the Graph



Approach taken by: TensorFlow XLA, Intel NGraph, Nvidia TensorRT



The Remaining Gap

Frameworks



CNTK

Computational Graph Optimization

need to build and optimize operators for each hardware,
variant of layout, precision, threading pattern ...

Hardware





Tensor Level Optimizations

Frameworks



CNTK

Computational Graph Optimization

Tensor Expression Language

```
C = t.compute((m, n),  
             lambda i, j: t.sum(A[i, k] * B[j, k], axis=k))
```

Hardware





Tensor Index Expression

Compute $C = \text{dot}(A, B.T)$

```
import tvn
```

```
m, n, h = tvn.var('m'), tvn.var('n'), tvn.var('h')
```

```
A = tvn.placeholder((m, h), name='A')
```

```
B = tvn.placeholder((n, h), name='B')
```

Inputs

```
k = tvn.reduce_axis((0, h), name='k')
```

```
C = tvn.compute((m, n), lambda i, j: tvn.sum(A[i, k] * B[j, k], axis=k))
```

Shape of C

Computation Rule



Tensor Expressions are Expressive

Affine Transformation

```
out = tvm.compute((n, m), lambda i, j: tvm.sum(data[i, k] * w[j, k], k))  
out = tvm.compute((n, m), lambda i, j: out[i, j] + bias[i])
```

Convolution

```
out = tvm.compute((c, h, w),  
    lambda i, x, y: tvm.sum(data[kc,x+kx,y+ky] * w[i,kx,ky], [kx,ky,kc]))
```

ReLU

```
out = tvm.compute(shape, lambda *i: tvm.max(0, out(*i)))
```




Emerging Tools Using Tensor Expression Language

Halide: Image processing language

Loopy: python based kernel generator

TACO: sparse tensor code generator

Tensor Comprehension



Schedule: Tensor Expression to Code

Tensor Expression Language

```
C = t.compute((m, n),  
             lambda i, j: t.sum(A[i, k] * B[j, k], axis=k))
```

Key Idea:
Separation of Compute
and Schedule
introduced by Halide

Schedule Optimizations

Hardware





Example Schedule Transformation

```
C = tvm.compute((n,), lambda i: A[i] + B[i])  
s = tvm.create_schedule(C.op)
```

```
for (int i = 0; i < n; ++i) {  
    C[i] = A[i] + B[i];  
}
```



Example Schedule Transformation

```
C = tvm.compute((n,), lambda i: A[i] + B[i])  
s = tvm.create_schedule(C.op)  
xo, xi = s[C].split(s[C].axis[0], factor=32)
```

```
for (int xo = 0; xo < ceil(n / 32); ++xo) {  
  for (int xi = 0; xi < 32; ++xi) {  
    int i = xo * 32 + xi;  
    if (i < n) {  
      C[i] = A[i] + B[i];  
    }  
  }  
}
```



Example Schedule Transformation

```
C = tvm.compute((n,), lambda i: A[i] + B[i])
s = tvm.create_schedule(C.op)
xo, xi = s[C].split(s[C].axis[0], factor=32)
s[C].recorder(xi, xo)
```

```
for (int xi = 0; xi < 32; ++xi) {
  for (int xo = 0; xo < ceil(n / 32); ++xo) {
    int i = xo * 32 + xi;
    if (i < n) {
      C[i] = A[i] + B[i];
    }
  }
}
```



Example Schedule Transformation

```
C = tvm.compute((n,), lambda i: A[i] + B[i])
s = tvm.create_schedule(C.op)
xo, xi = s[C].split(s[C].axis[0], factor=32)
s[C].recorder(xi, xo)
s[C].bind(xo, tvm.thread_axis("blockIdx.x"))
s[C].bind(xi, tvm.thread_axis("threadIdx.x"))
```

```
int i = threadIdx.x * 32 + blockIdx.x;
if (i < n) {
    C[i] = A[i] + B[i];
}
```



Key Challenge: Good Space of Schedule

Should contain any knobs that produces a logically equivalent program that runs well on backend models

Must contain the common manual optimization patterns

Need to actively evolve to incorporate new techniques

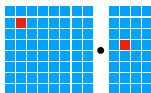


Hardware-aware Search Space

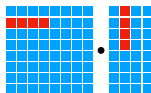
CPU



Compute Primitives



scalar



vector

Memory Subsystem



implicitly managed

Loop Transformations

Cache Locality

Vectorization

Reuse primitives from prior work:
Halide, Loopy



Challenge to Support Diverse Hardware Backends

CPU



GPU



TPU-like specialized Accelerators



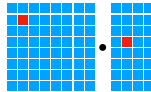


Hardware-aware Search Space

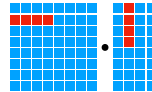
GPUs



Compute Primitives

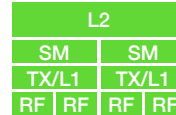


scalar



vector

Memory Subsystem



mixed

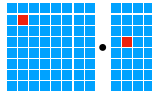


Hardware-aware Search Space

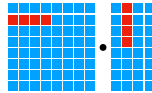
GPUs



Compute Primitives

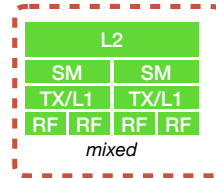


scalar



vector

Memory Subsystem



Shared memory among
compute cores

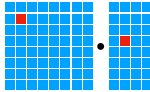


Hardware-aware Search Space

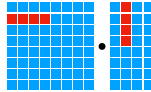
GPUs



Compute Primitives

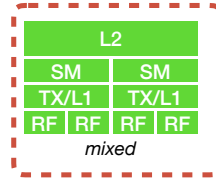


scalar



vector

Memory Subsystem



Shared memory among
compute cores

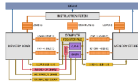
Use of Shared
Memory

Thread
Cooperation

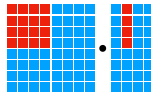


Hardware-aware Search Space

TPU-like Specialized Accelerators



Compute Primitives



tensor

Memory Subsystem

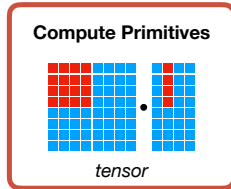
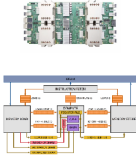


explicitly managed

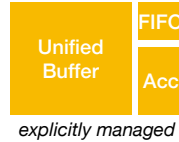


Hardware-aware Search Space

TPU-like Specialized Accelerators



Memory Subsystem





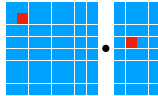
Tensorization Challenge

**Compute
primitives**



Tensorization Challenge

Compute
primitives

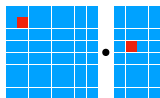


scalar

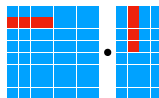


Tensorization Challenge

Compute
primitives



scalar

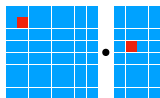


vector

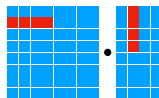


Tensorization Challenge

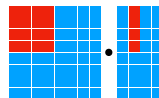
Compute
primitives



scalar



vector

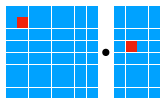


tensor

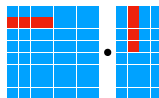


Tensorization Challenge

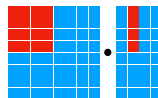
Compute primitives



scalar



vector



tensor

Hardware designer:
declare tensor instruction interface
with Tensor Expression

```
w, x = t.placeholder((8, 8)), t.placeholder((8, 8))
k = t.reduce_axis((0, 8))
y = t.compute((8, 8), lambda i, j:
    t.sum(w[i, k] * x[j, k], axis=k))
```

← declare behavior

```
def gemm_intrin_lower(inputs, outputs):
    ww_ptr = inputs[0].access_ptr("r")
    xx_ptr = inputs[1].access_ptr("r")
    zz_ptr = outputs[0].access_ptr("w")
    compute = t.hardware_intrin("gemm8x8", ww_ptr, xx_ptr, zz_ptr)
    reset = t.hardware_intrin("fill_zero", zz_ptr)
    update = t.hardware_intrin("fuse_gemm8x8_add", ww_ptr, xx_ptr, zz_ptr)
    return compute, reset, update
```

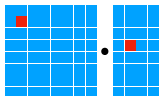
← lowering rule to generate hardware intrinsics to carry out the computation

```
gemm8x8 = t.decl_tensor_intrin(y.op, gemm_intrin_lower)
```

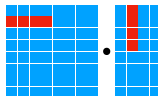


Tensorization Challenge

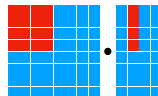
Compute primitives



scalar



vector



tensor

Hardware designer:
declare tensor instruction interface
with Tensor Expression

```
w, x = t.placeholder((8, 8)), t.placeholder((8, 8))
k = t.reduce_axis((0, 8))
y = t.compute((8, 8), lambda i, j:
    t.sum(w[i, k] * x[j, k], axis=k))
```

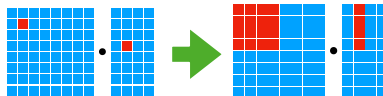
declare behavior

```
def gemm_intrin_lower(inputs, outputs):
    ww_ptr = inputs[0].access_ptr("r")
    xx_ptr = inputs[1].access_ptr("r")
    zz_ptr = outputs[0].access_ptr("w")
    compute = t.hardware_intrin("gemm8x8", ww_ptr, xx_ptr, zz_ptr)
    reset = t.hardware_intrin("fill_zero", zz_ptr)
    update = t.hardware_intrin("fuse_gemm8x8_add", ww_ptr, xx_ptr, zz_ptr)
    return compute, reset, update
```

lowering rule to generate
hardware intrinsics to carry
out the computation

```
gemm8x8 = t.decl_tensor_intrin(y.op, gemm_intrin_lower)
```

Tensorize:
transform program
to use tensor instructions



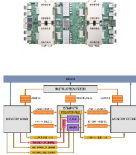
scalar

tensor

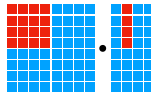


Hardware-aware Search Space

TPU-like Specialized Accelerators



Compute Primitives



tensor

Memory Subsystem

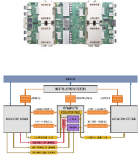


explicitly managed

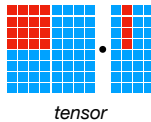


Hardware-aware Search Space

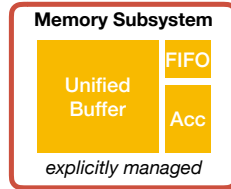
TPU-like Specialized Accelerators



Compute Primitives



tensor





Software Support for Latency Hiding

Single Module
No Task-Pipelining

load
inputs

load
weights

matrix
multiplication

store
outputs

load
inputs

load
weights

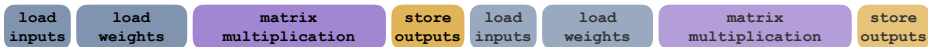
matrix
multiplication

store
outputs

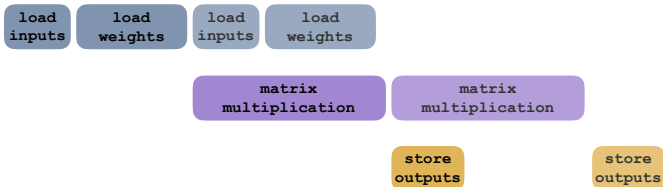


Software Support for Latency Hiding

Single Module
No Task-Pipelining



Multiple-Module
Task-Level Pipelining



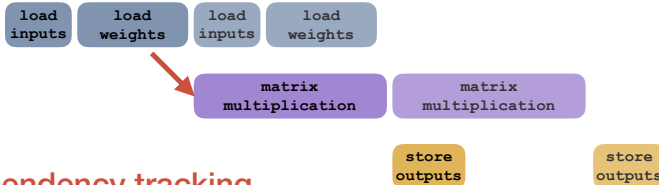


Software Support for Latency Hiding

Single Module
No Task-Pipelining



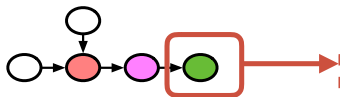
Multiple-Module
Task-Level Pipelining



**Explicit dependency tracking
managed by software to hide memory latency**



Hardware-aware Search Space



Tensor Expression Language

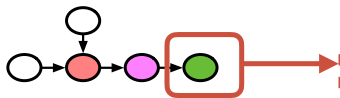
```
C = tvm.compute((m, n),  
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Hardware





Hardware-aware Search Space



Tensor Expression Language

```
C = tvm.compute((m, n),  
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Primitives in prior work:
Halide, Loopy

Loop
Transformations

Thread
Bindings

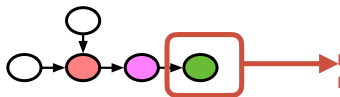
Cache
Locality

Hardware





Hardware-aware Search Space



Tensor Expression Language

```
C = tvn.compute((m, n),  
    lambda y, x: tvn.sum(A[k, y] * B[k, x], axis=k))
```

Primitives in prior work:
Halide, Loopy

Loop Transformations

Thread Bindings

Cache Locality

New primitives for GPUs,
and enable TPU-like
Accelerators

Thread Cooperation

Tensorization

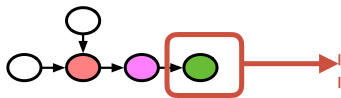
Latency Hiding

Hardware





Hardware-aware Search Space



Tensor Expression Language

```
C = tvm.compute((m, n),  
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Loop Transformations

Thread Bindings

Cache Locality

Thread Cooperation

Tensorization

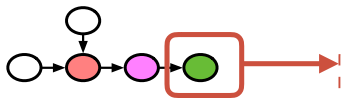
Latency Hiding

Hardware





Hardware-aware Search Space



Tensor Expression Language

```
C = tvm.compute((m, n),  
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Loop Transformations

Thread Bindings

Cache Locality

Thread Cooperation

Tensorization

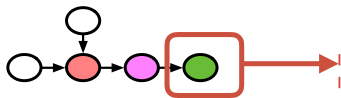
Latency Hiding

Hardware





Hardware-aware Search Space



Tensor Expression Language

```
C = tvm.compute((m, n),  
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Billions
of possible
optimization
choices

Loop
Transformations

Thread
Bindings

Cache
Locality

Thread
Cooperation

Tensorization

Latency
Hiding

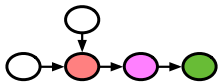
Hardware





Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

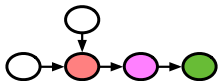
Hardware





Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

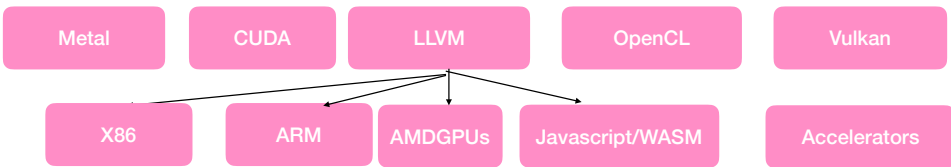
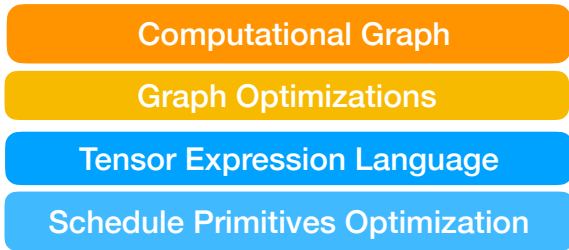
Machine Learning based Program Optimizer

Hardware





Global View of TVM Stack



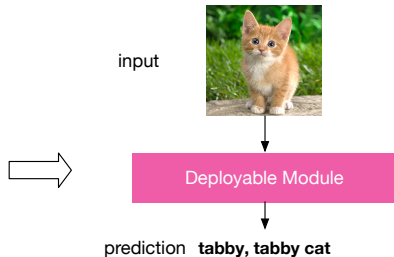


High Level Compilation Frontend

```
import tvm
import nnvm.frontend
import nnvm.compiler
```

```
graph, params =
nnvm.frontend.from_keras(keras_resnet50)
graph, lib, params =
nnvm.compiler.build(graph, target)
```

```
module = runtime.create(graph, lib, tvm.gpu(0))
module.set_input(**params)
module.run(data=data_array)
output = tvm.nd.empty(out_shape, ctx=tvm.gpu(0))
module.get_output(0, output)
```



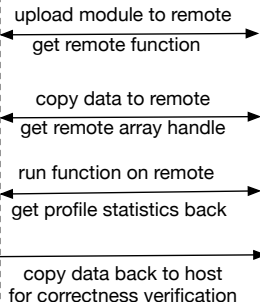
On languages and platforms you choose





Program Your Phone with Python from Your Laptop

RPC Server on
Embedded Device

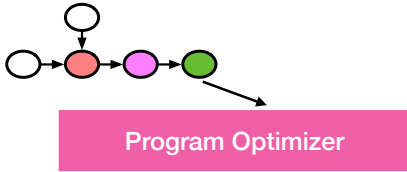


Compiler Stack

```
lib = t.build(s, [A, B],  
             'llvm -target=armv7l-none-linux-gnueabihf',  
             name='myfunc')  
remote = t.rpc.connect(host, port)  
lib.save('myfunc.o')  
remote.upload('myfunc.o')  
f = remote.load_module('myfunc.o')  
ctx = remote.cpu(0)  
a = t.nd.array(np.random.uniform(size=1024), ctx)  
b = t.nd.array(np.zeros(1024), ctx)  
remote_timer = f.time_evaluator('myfunc', ctx, number=10)  
time_cost = remote_timer(a, b)  
  
np.testing.assert_equal(b.asnumpy(), expected)
```

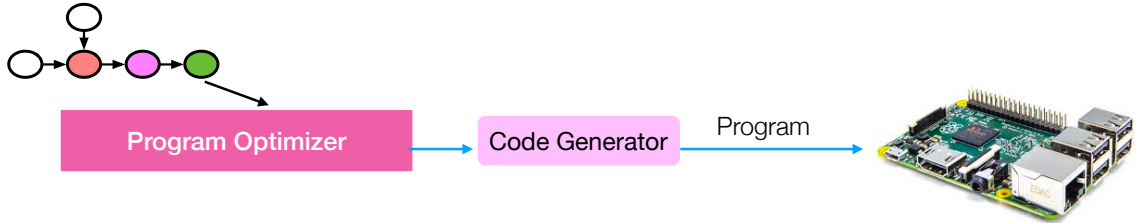


Learning-based Program Optimizer



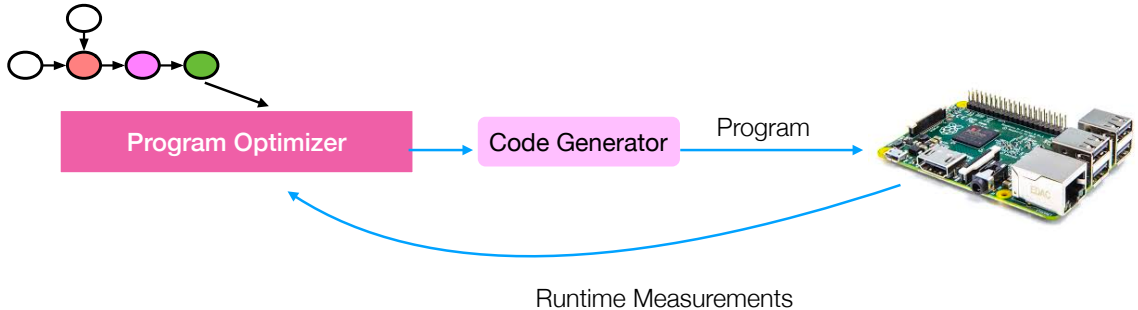


Learning-based Program Optimizer



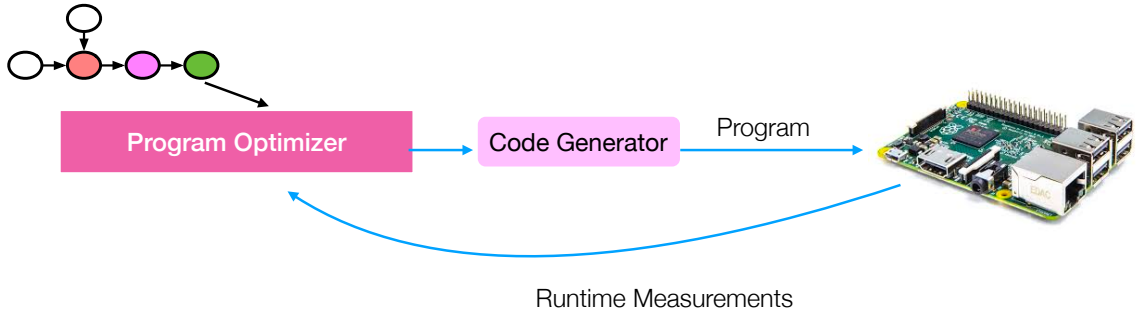


Learning-based Program Optimizer





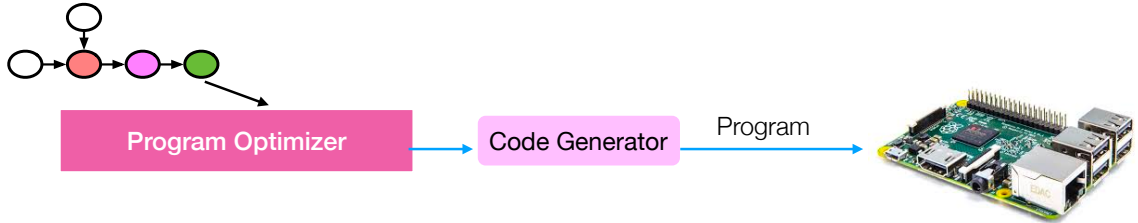
Learning-based Program Optimizer



High experiment cost,
each trial costs ~1second

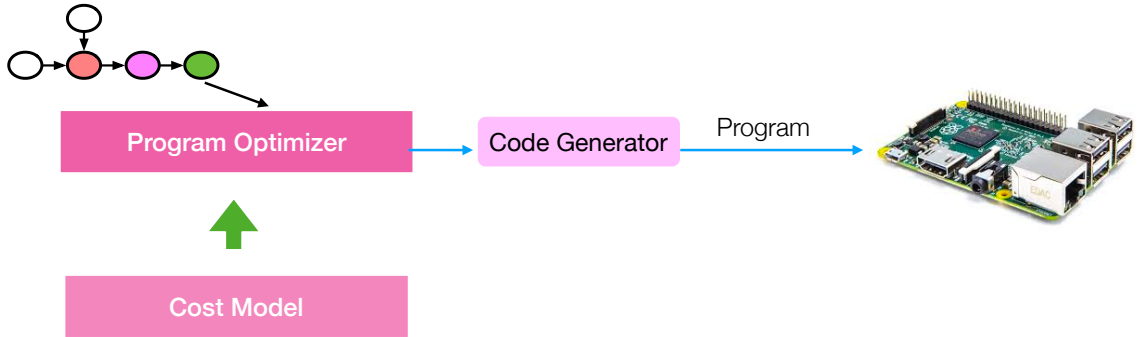


Learning-based Program Optimizer



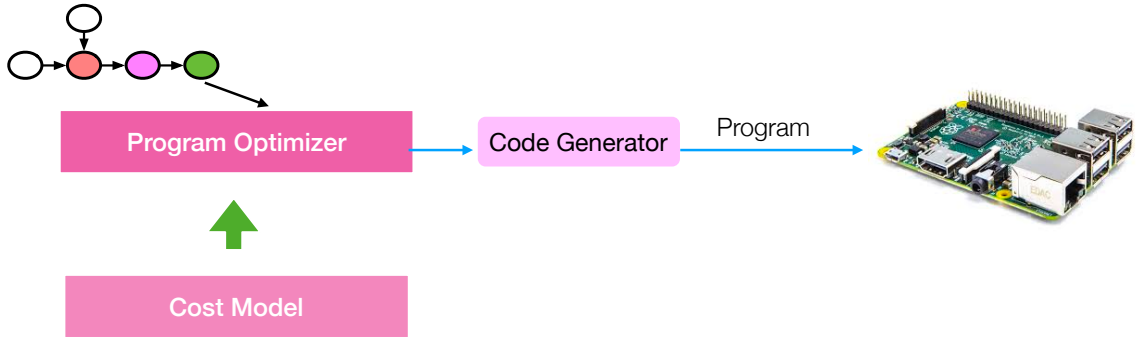


Learning-based Program Optimizer





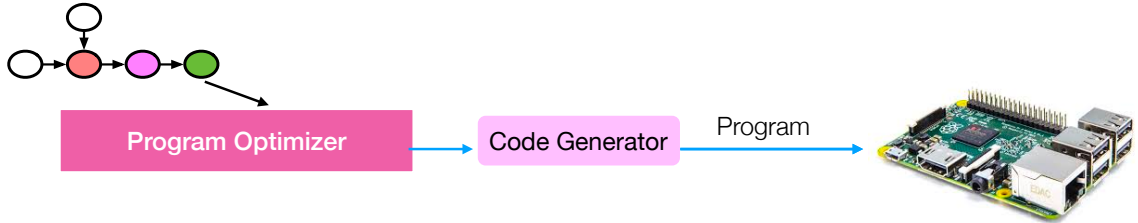
Learning-based Program Optimizer



Need reliable cost model per hardware

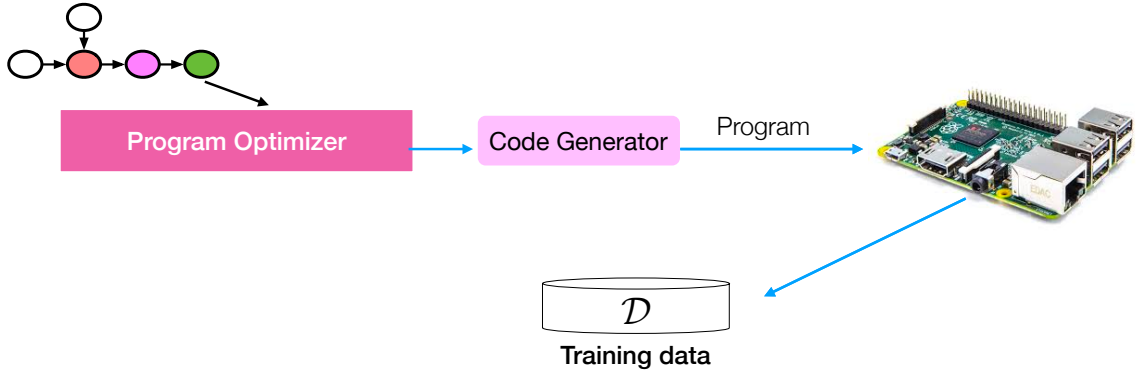


Learning-based Program Optimizer



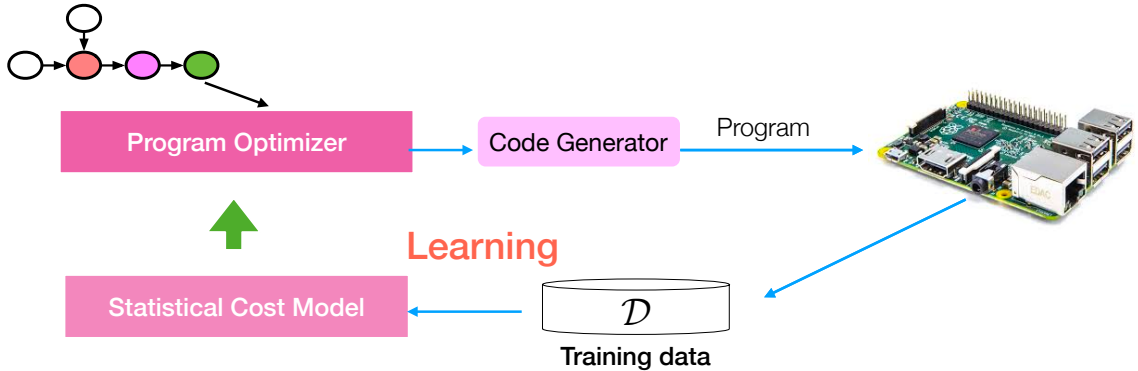


Learning-based Program Optimizer



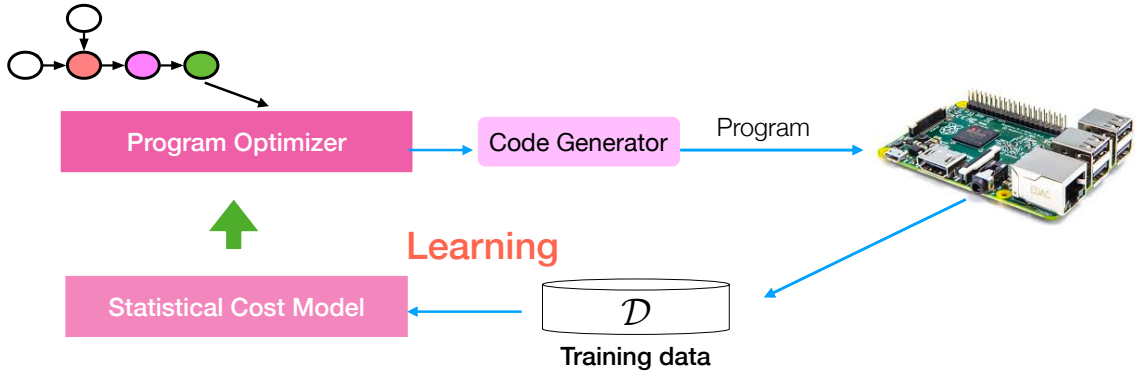


Learning-based Program Optimizer





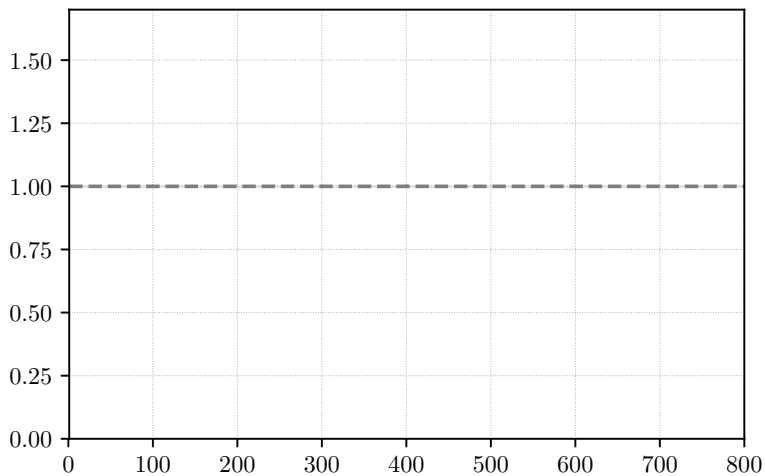
Learning-based Program Optimizer



Adapt to hardware type by learning
Make prediction in 1ms level

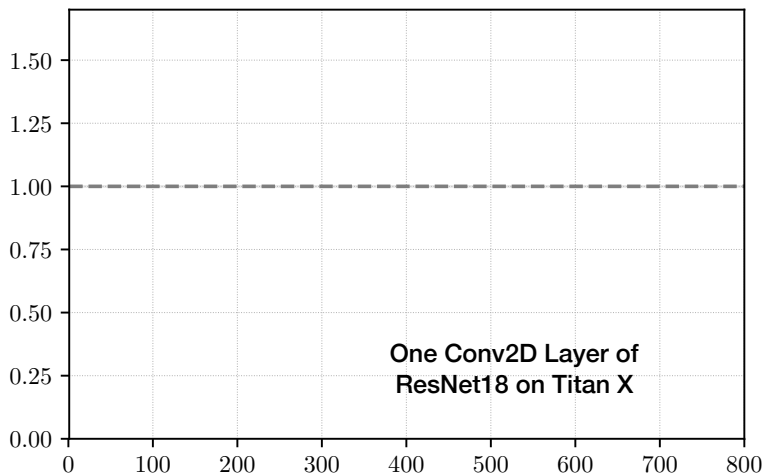


Effectiveness of ML based Model



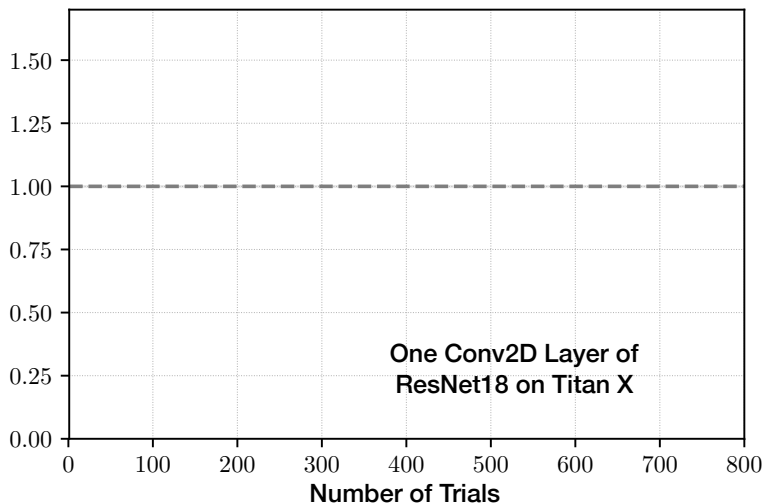


Effectiveness of ML based Model



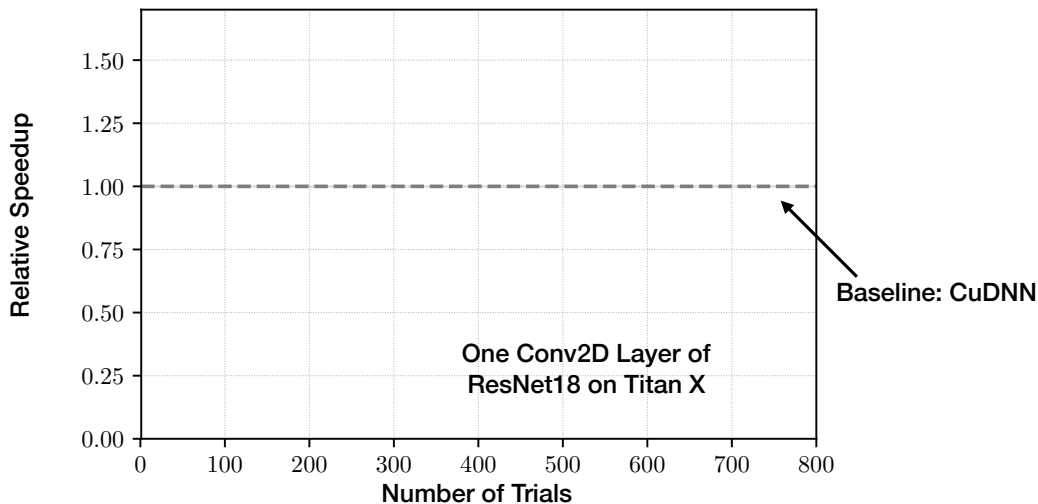


Effectiveness of ML based Model



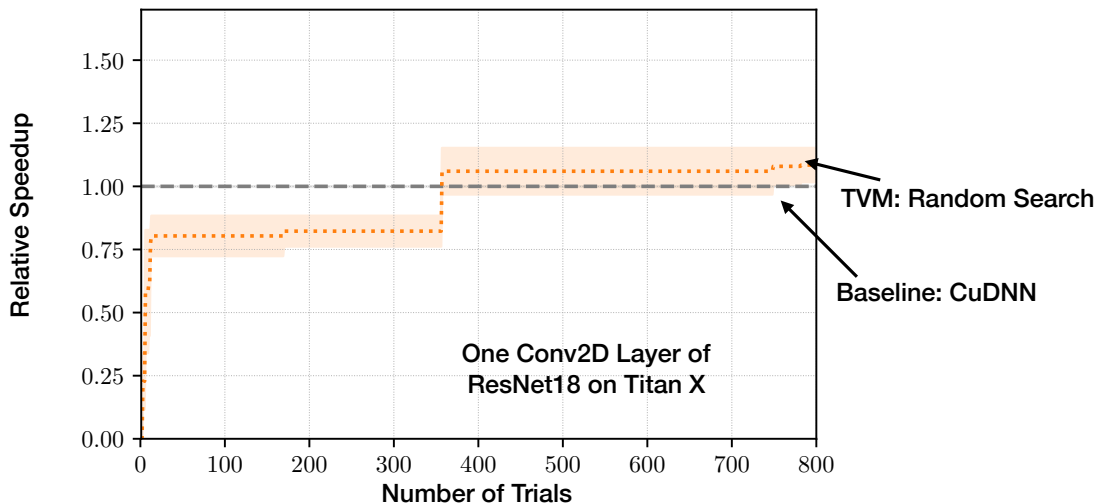


Effectiveness of ML based Model



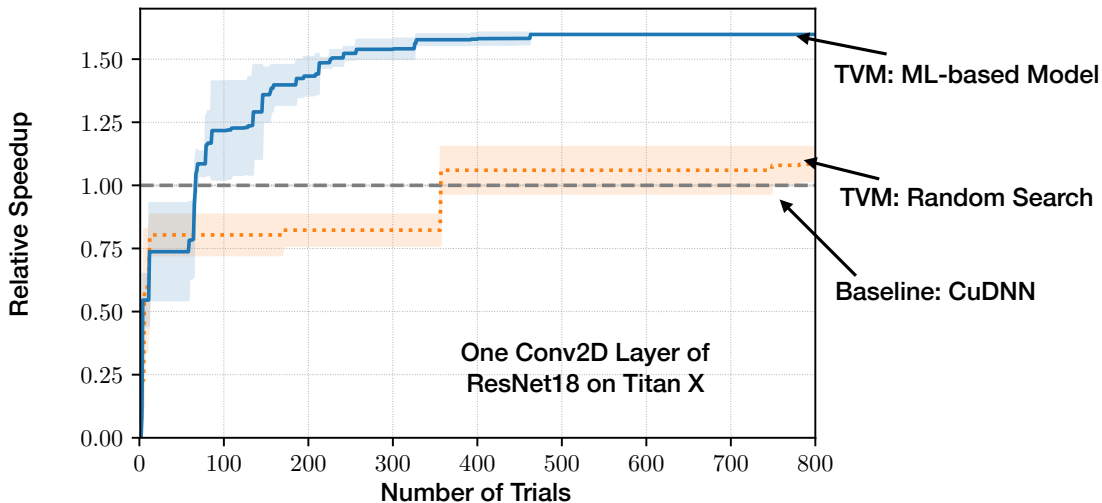


Effectiveness of ML based Model





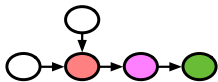
Effectiveness of ML based Model





Learning-based Learning System

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

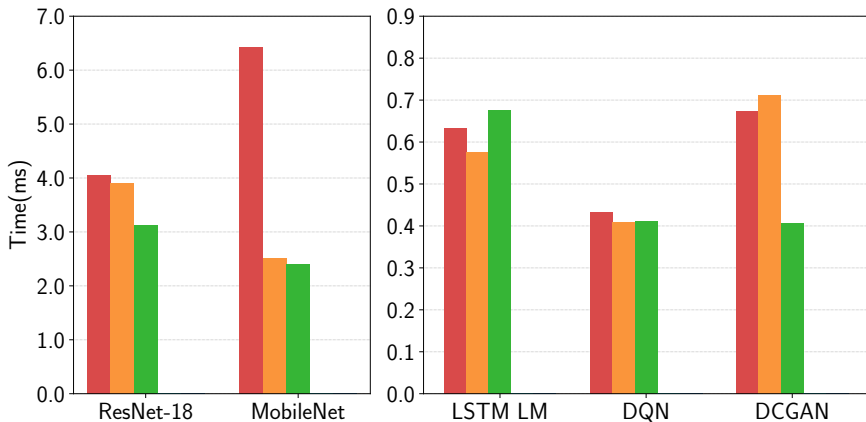
Hardware





End to End Inference Performance (Nvidia Titan X)

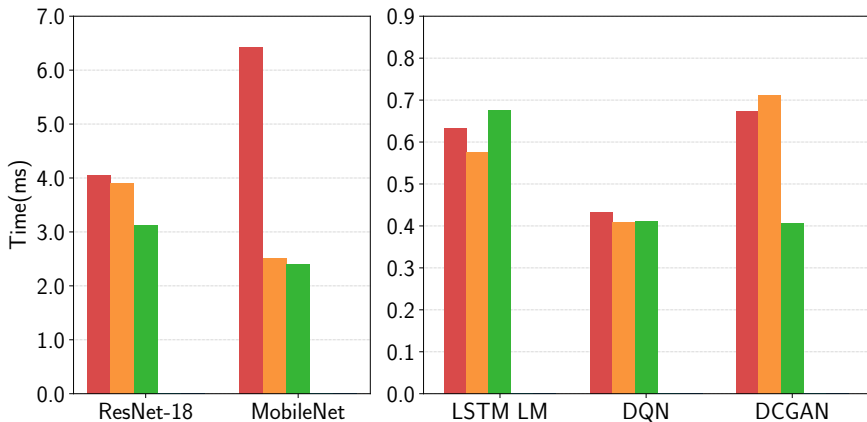
Tensorflow Apache MXNet
Tensorflow-XLA





End to End Inference Performance (Nvidia Titan X)

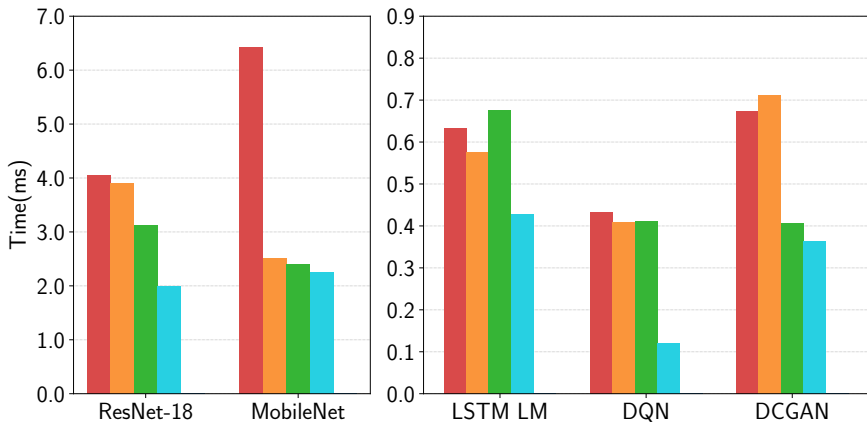
Backed by cuDNN





End to End Inference Performance (Nvidia Titan X)

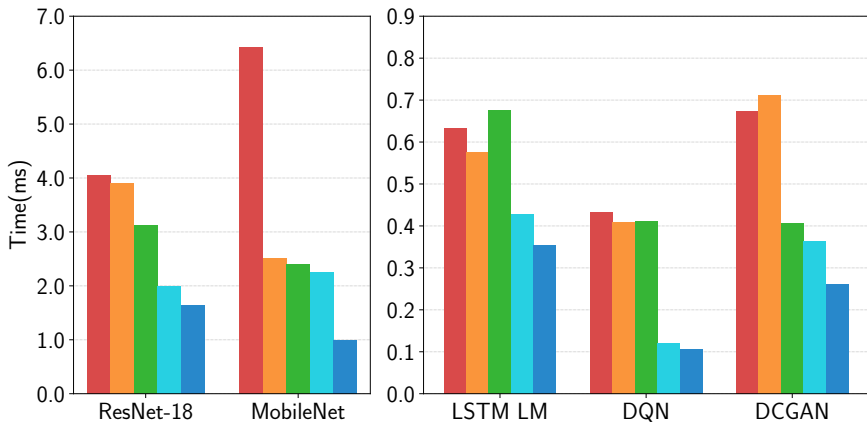
Tensorflow Apache MXNet TVM: without graph optimizations
Tensorflow-XLA





End to End Inference Performance (Nvidia Titan X)

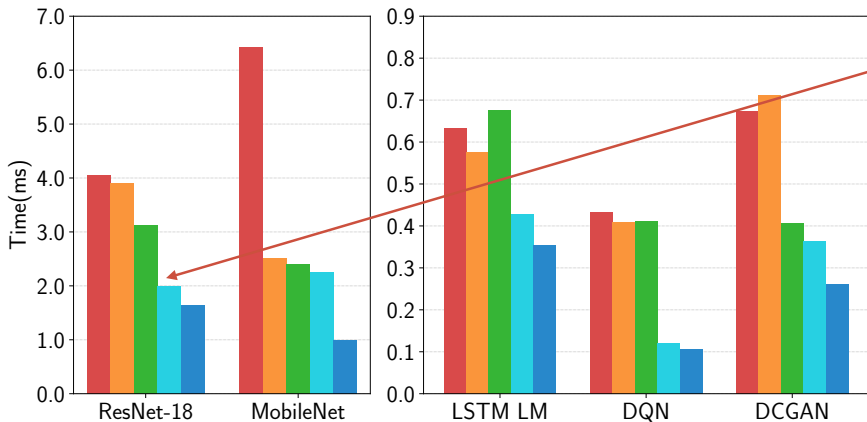
Tensorflow Apache MXNet TVM: without graph optimizations
Tensorflow-XLA TVM: all optimizations





End to End Inference Performance (Nvidia Titan X)

Tensorflow Apache MXNet TVM: without graph optimizations
Tensorflow-XLA TVM: all optimizations

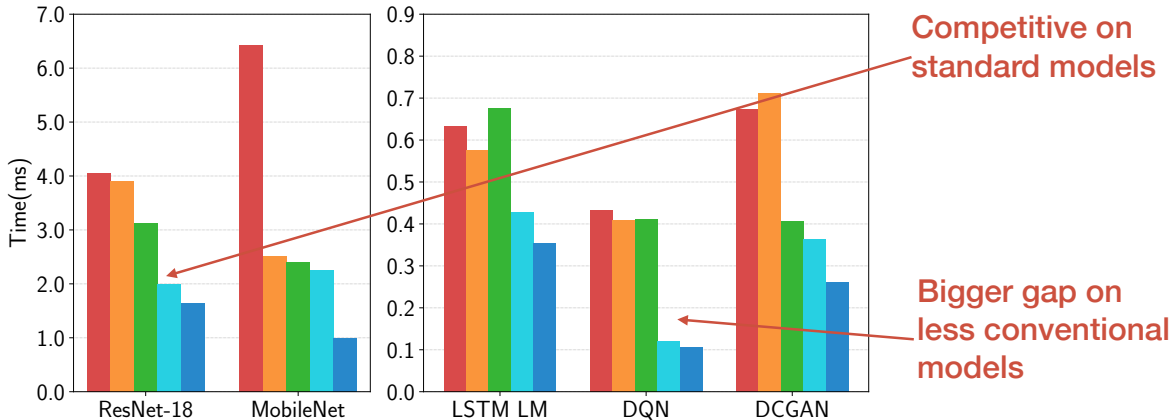


Competitive on standard models



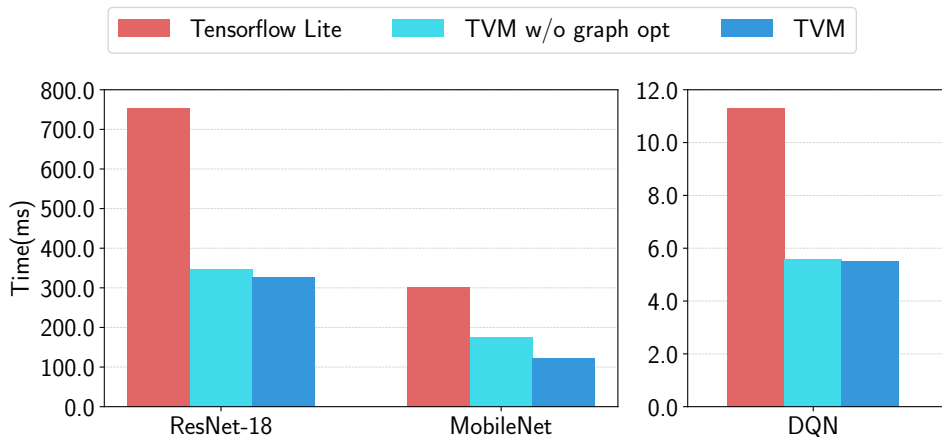
End to End Inference Performance (Nvidia Titan X)

Tensorflow Apache MXNet TVM: without graph optimizations
Tensorflow-XLA TVM: all optimizations





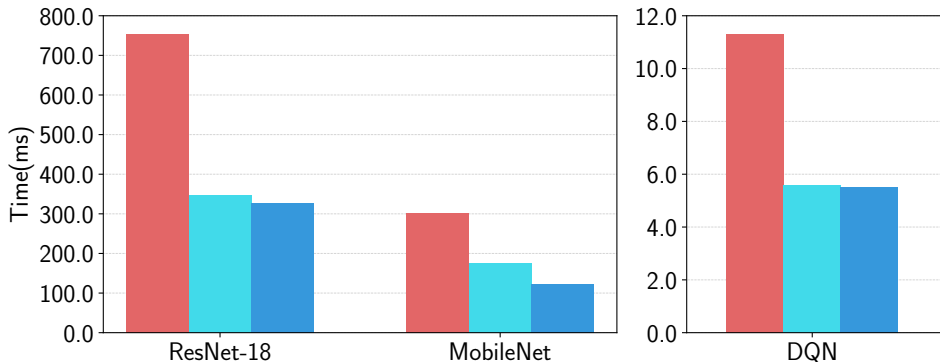
End to End Performance(ARM Cortex-A53)





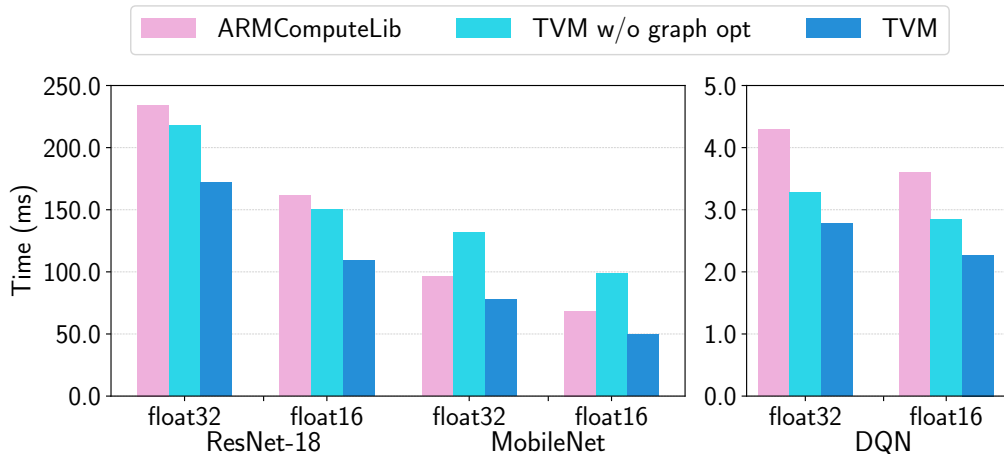
End to End Performance(ARM Cortex-A53)

Specially optimized for
Embedded system(ARM)





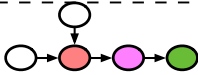
End to End Performance(ARM GPU)





Supporting New Specialized Accelerators

Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer

LLVM

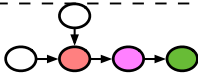
CUDA





Supporting New Specialized Accelerators

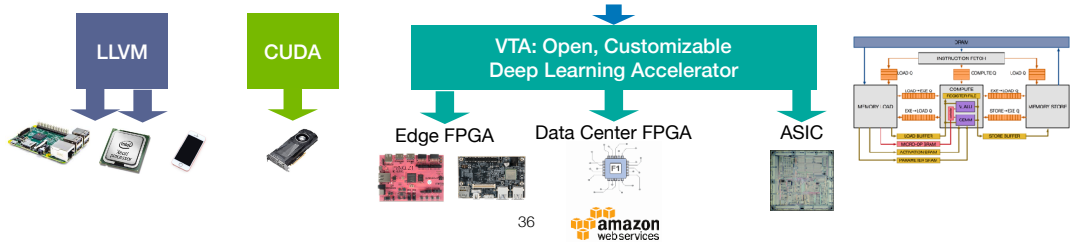
Frameworks



High-level data flow graph and optimizations

Hardware aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer





TVM/MTA: Full Stack Open Source System



High-level Optimizations

Tensor Program Search Space

ML-based Optimizer





TVM/VTA: Full Stack Open Source System



High-level Optimizations

Tensor Program Search Space

ML-based Optimizer

VTA MicroArchitecture





TVM/VTA: Full Stack Open Source System



High-level Optimizations

Tensor Program Search Space

ML-based Optimizer

VTA Hardware/Software Interface (ISA)

VTA MicroArchitecture





TVM/VTA: Full Stack Open Source System



High-level Optimizations

Tensor Program Search Space

ML-based Optimizer

VTA Runtime & JIT Compiler

VTA Hardware/Software Interface (ISA)

VTA MicroArchitecture





TVM/VTA: Full Stack Open Source System



High-level Optimizations

Tensor Program Search Space

ML-based Optimizer

VTA Runtime & JIT Compiler

VTA Hardware/Software Interface (ISA)

VTA MicroArchitecture

VTA Simulator





TVM/VTA: Full Stack Open Source System



High-level Optimizations

Tensor Program Search Space

ML-based Optimizer

VTA Runtime & JIT Compiler

VTA Hardware/Software Interface (ISA)

VTA MicroArchitecture

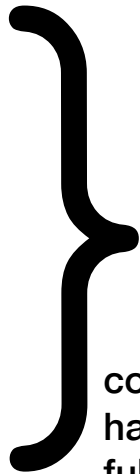
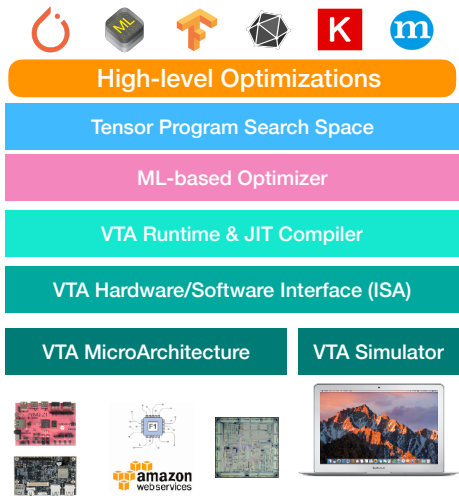
VTA Simulator



- JIT compile accelerator micro code
- Support heterogenous devices, 10x better than CPU on the same board.
- Move hardware complexity to software



TVM/VTA: Full Stack Open Source System

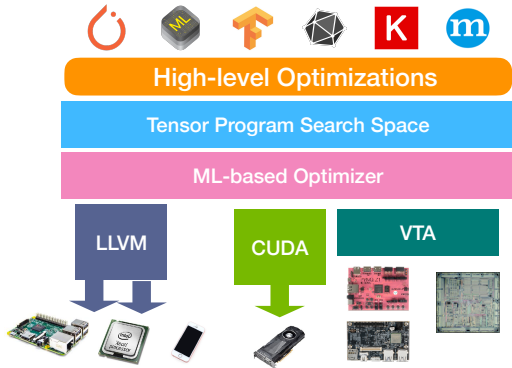


- JIT compile accelerator micro code
- Support heterogenous devices, 10x better than CPU on the same board.
- Move hardware complexity to software

**compiler, driver,
hardware design
full stack open source**



TVM: Learning-based Learning System



Check it out!

