2020 embedded VISION summit

Introduction to the TVM Open Source Deep Learning Compiler Stack

Luis Ceze

w/ Tianqi Chen, Thierry Moreau, Jared Roesch, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Chien-Yu Lin, Haichen Shen, Leyuan Wang, Yuwei Hu, Carlos Guestrin, Arvind Krishnamurthy, Zach Tatlock, and many in the Apache TVM community!

tvm.ai

W PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

OctoML

sampl

A perfect storm

Growing set of requirements: **Cost, latency, power, security & privacy**

Cambrian explosion of models, workloads, and use cases

CNN **GAN** **RNN** **MLP** **DQNN**

Rapidly evolving ML software ecosystem

Silicon scaling limitations (Dennard and Moore)

Cambrian explosion of HW backends. Heterogeneous HW

intel NVIDIA XILINX Microsoft QUALCOMM amazon Google HUAWEI

Current Dominant Deep Learning Systems Landscape

Orchestrators: Kubeflow, SELDON, ALGORITHMIA, Azure ML, GCP Datalab

Frameworks and Inference engines: ONNX RUNTIME, TensorRT

DL Compilers: nGraph, GLOW, MLIR

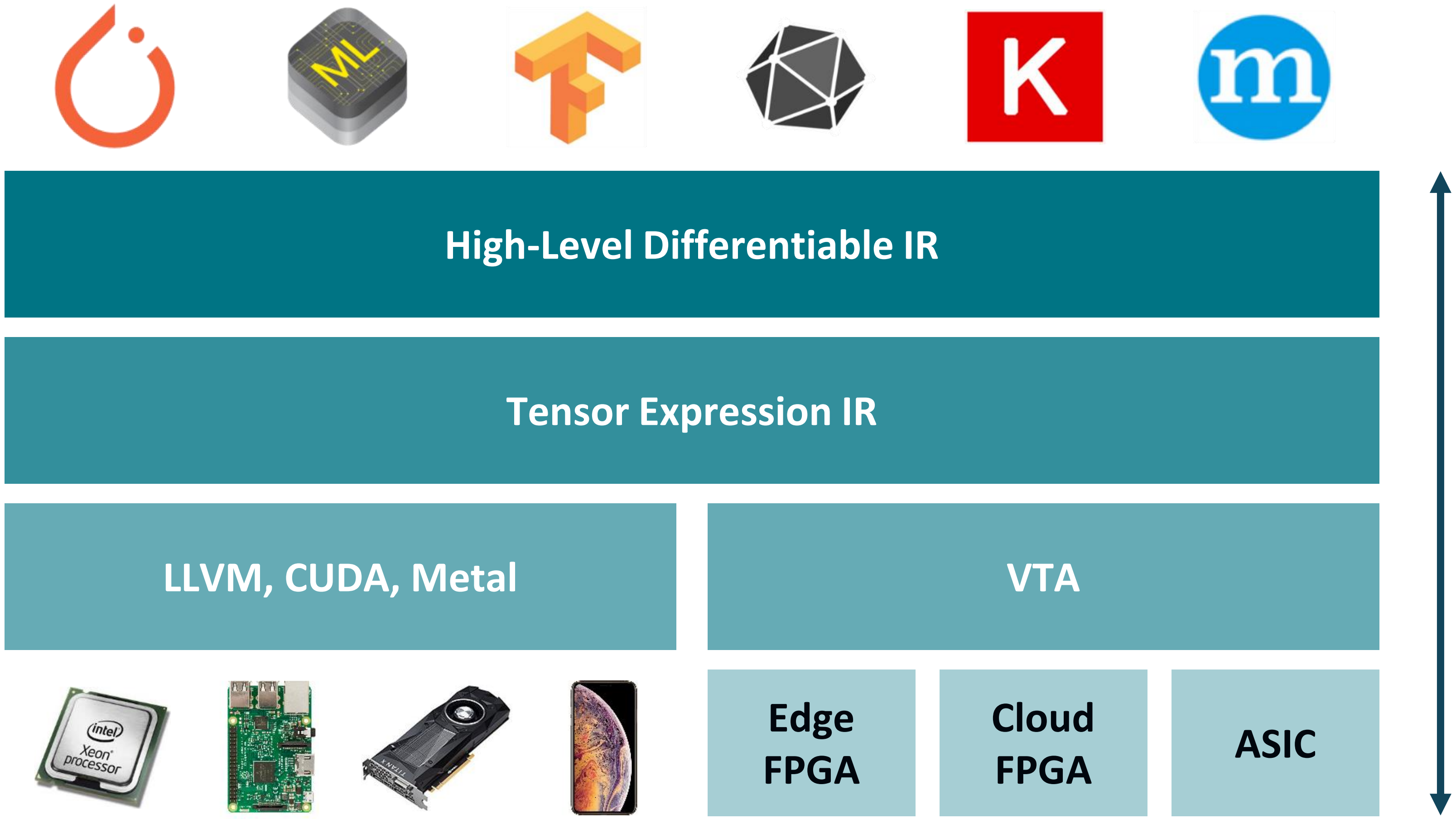
Kernel Libraries: cuDNN, NNPack, MKL-DNN

Hand optimized ↔ Open source, **automated** end-to-end optimization framework for deep learning

tvm

Hardware

tvm Stack



High-Level Differentiable IR

Tensor Expression IR

LLVM, CUDA, Metal

VTA

Edge FPGA

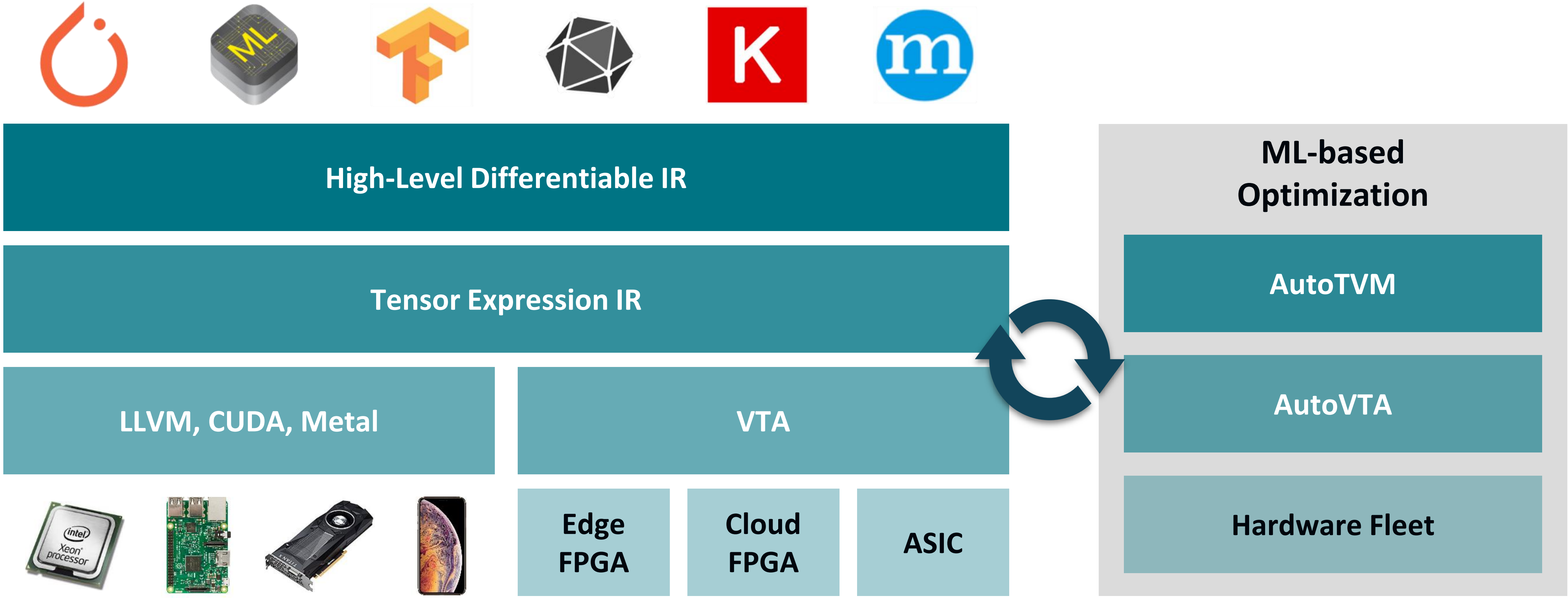
Cloud FPGA

ASIC

Open source synthesizable deep learning accelerator design

**End-to-end,
framework to metal open stack.
Research and deployment.**

Automated by Machine Learning



High-Level Differentiable IR

Tensor Expression IR

LLVM, CUDA, Metal

VTA

Edge FPGA

Cloud FPGA

ASIC

ML-based Optimization

AutoTVM

AutoVTA

Hardware Fleet

TVM: Automated End-to-end Optimizations for Deep Learning. Chen et al. OSDI 18

End-user perspective: Compile & deploy

Compile

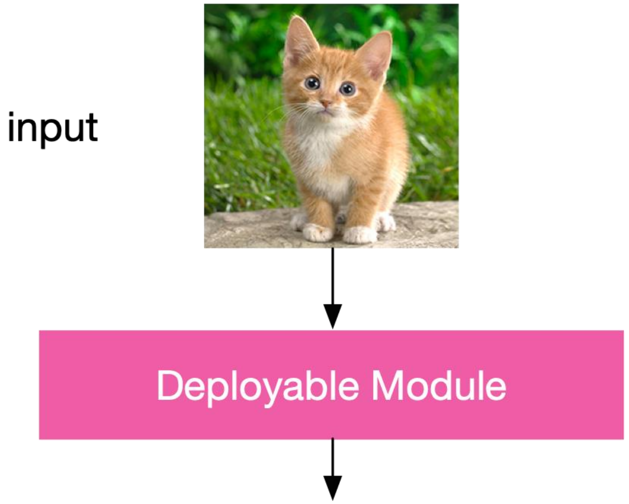
```
import tvm
from tvm import relay

graph, params =
Frontend.from_keras
(keras_resnet50)

graph, lib, params =
Relay.build(graph, target)
```

Deploy

```
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)
```



prediction **tabby, tabby cat**

Open Source Community and Impact

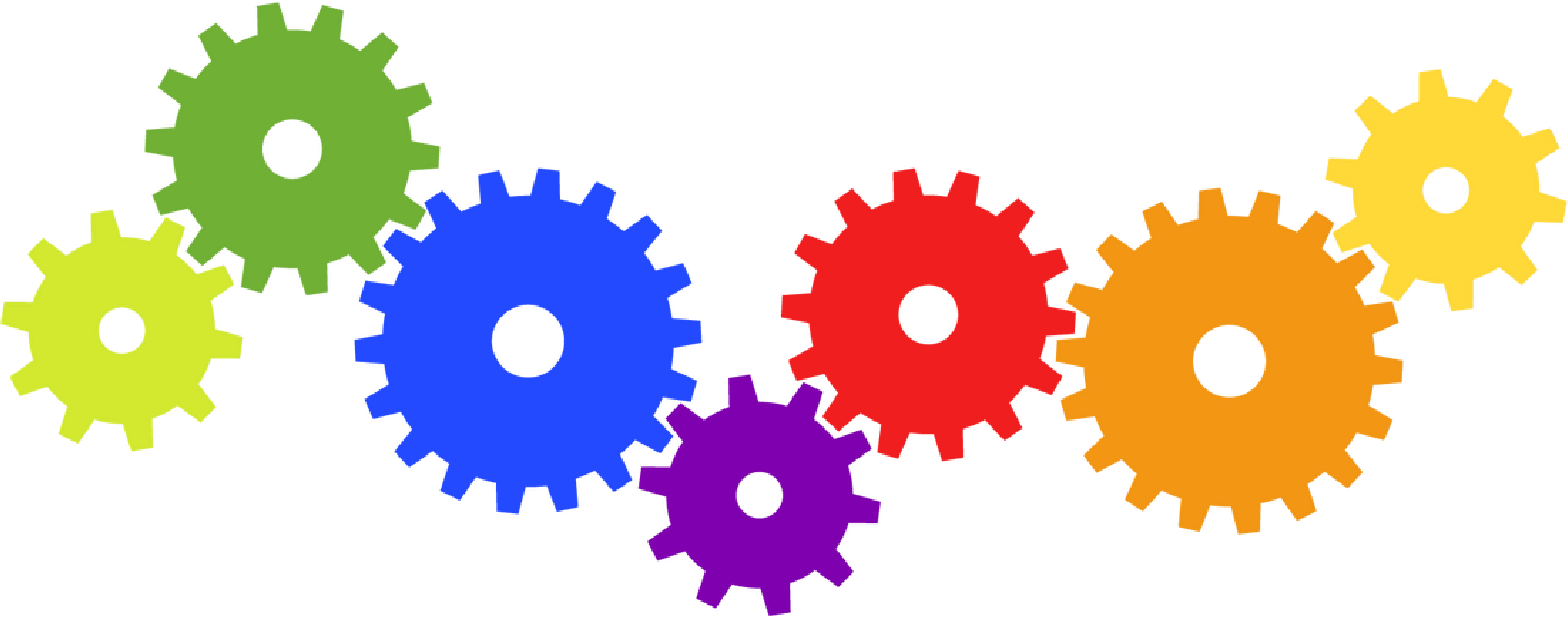
Open source: ~420+ contributors from UW, Berkeley, Cornell, UCLA, Amazon, Huawei, NTT, Facebook, Microsoft, Qualcomm, Alibaba, Intel, …

Used in production at leading companies

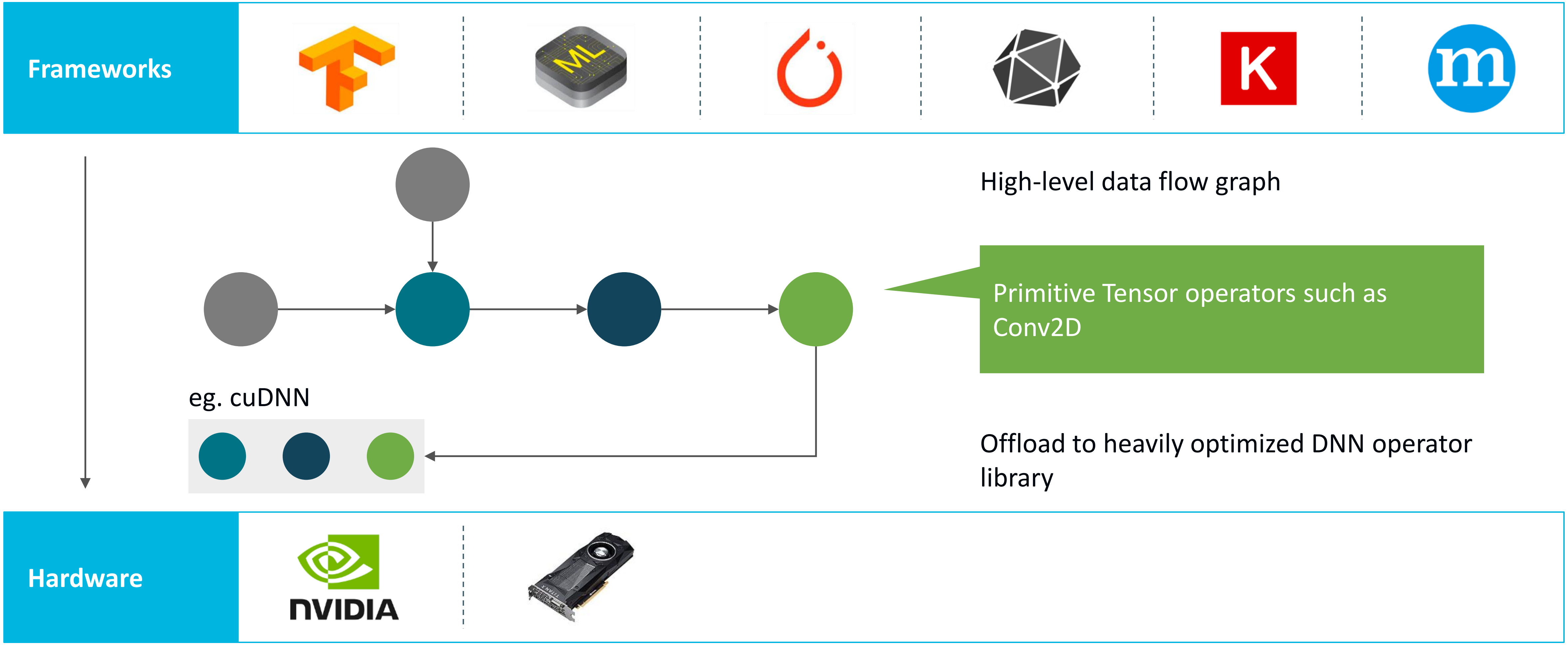
aws	Qualcomm	Facebook	Microsoft
Deep Learning Compiler Service	**DSP/Tensor engine for mobile**	**Mobile and Server Optimizations**	**Cloud-side model optimization**

THE APACHE® SOFTWARE FOUNDATION

Incubated as Apache TVM. Independent governance, allowing competitors to collaborate.



Existing Deep Learning Frameworks



Frameworks

High-level data flow graph

Primitive Tensor operators such as Conv2D

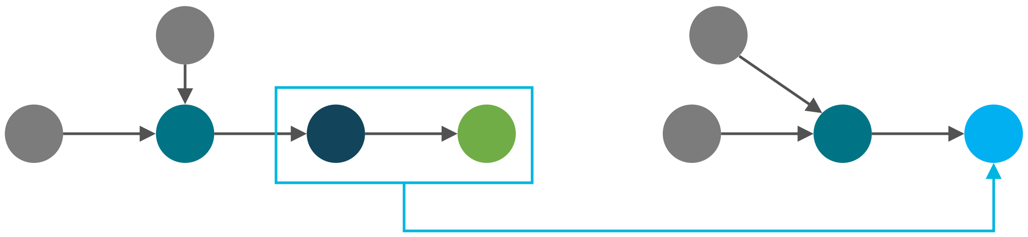
eg. cuDNN

Offload to heavily optimized DNN operator library

Hardware

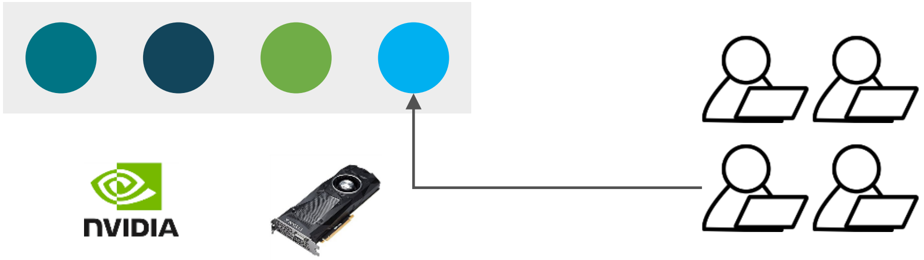
Engineering costs limits progress

Frameworks

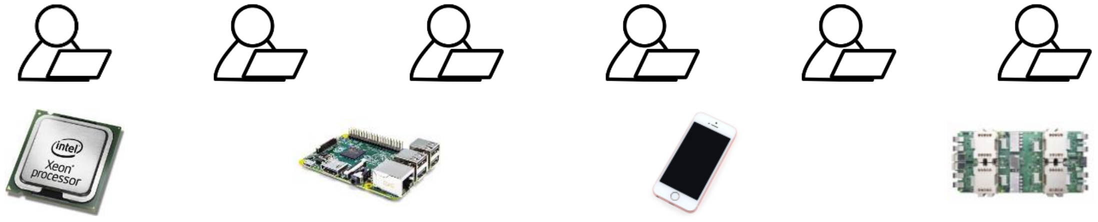


New operator introduced by operator fusion optimization potential benefit: 1.5x speedup

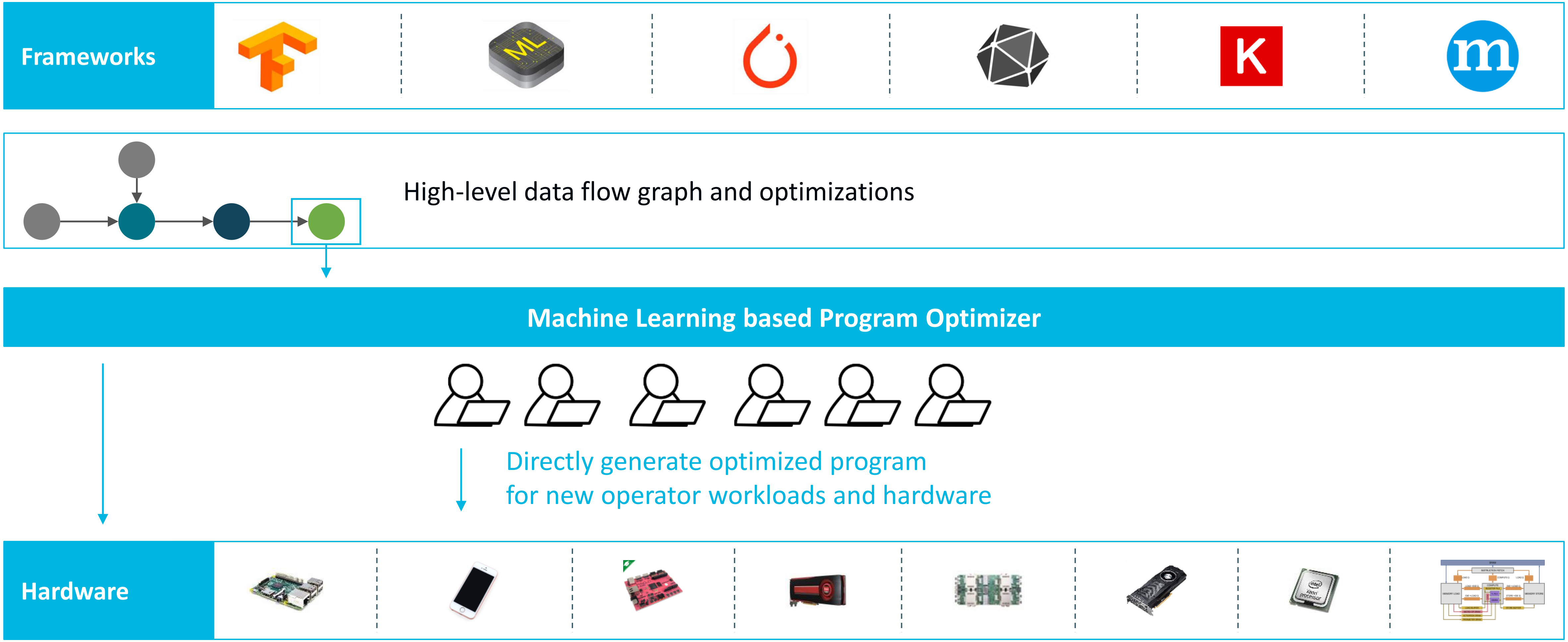
cuDNN



Engineering intensive



Our approach: Learning-based Learning System



Frameworks

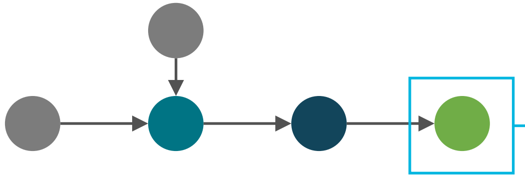
High-level data flow graph and optimizations

Machine Learning based Program Optimizer

Directly generate optimized program
for new operator workloads and hardware

Hardware

Tensor Compilation/Optimization as a search problem

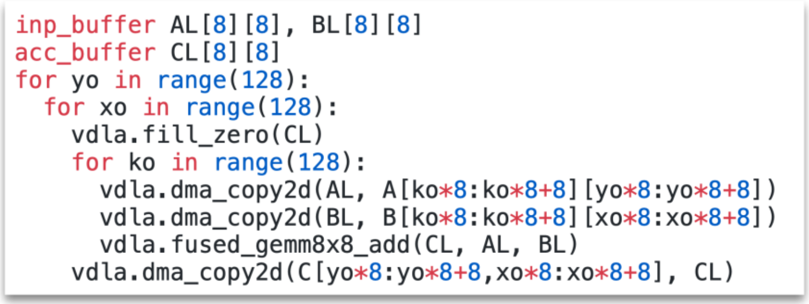


Tensor Expression (Specification)

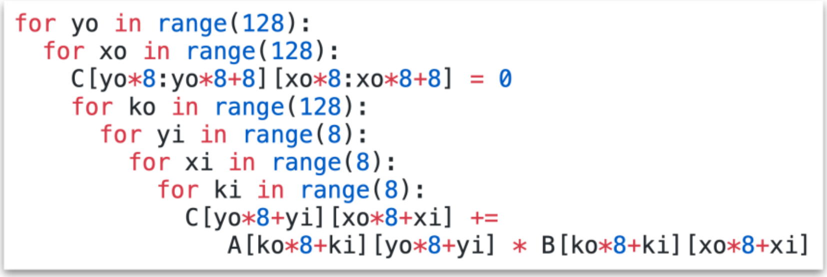
C = tvm.compute((m, n),
lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))

Search Space of Possible Program Optimizations

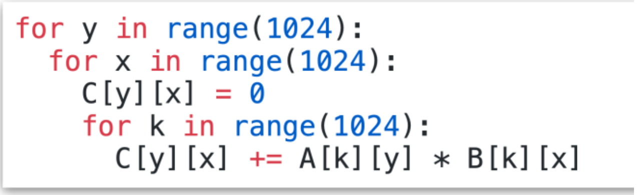
Low-level Program Variants



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

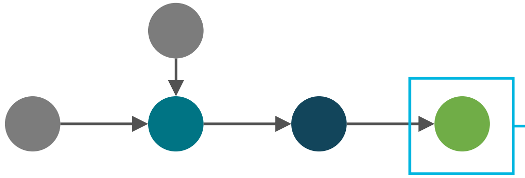


```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
               A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```



```
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]
```

Search Space Example (1/3)



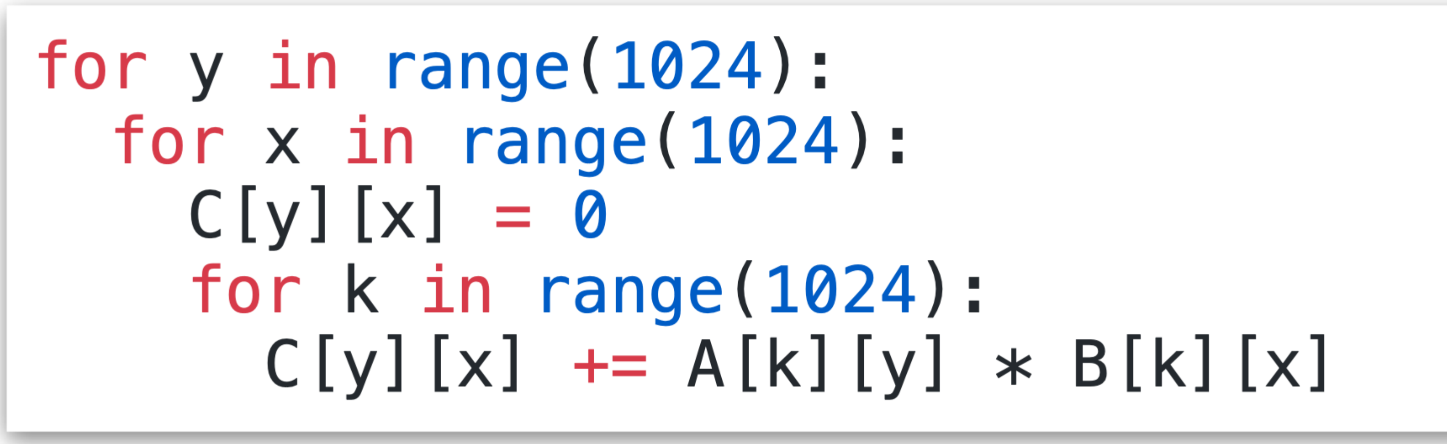
Tensor Expression (Specification)

C = tvm.compute((m, n),

lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))

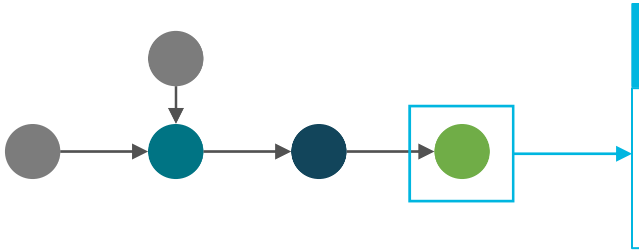
Search Space of Possible Program Optimizations

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]
```

Search Space Example (2/3)



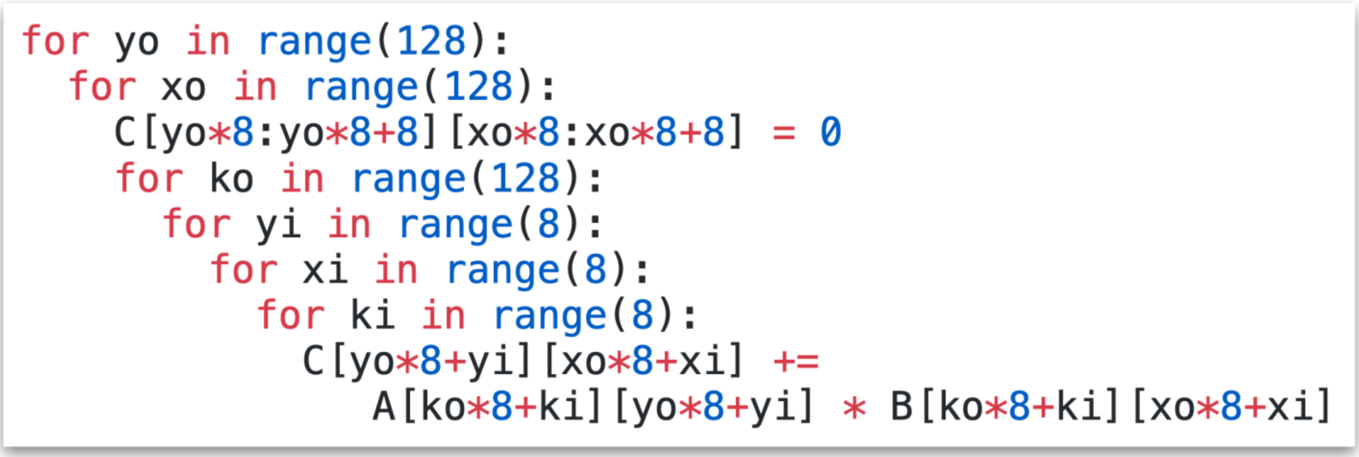
Tensor Expression (Specification)

C = tvm.compute((m, n),

lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))

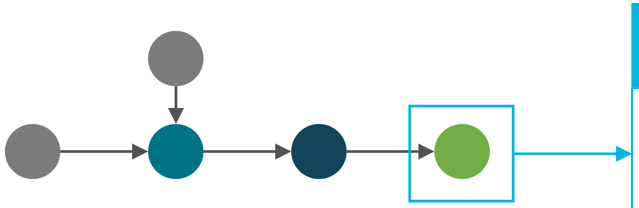
Search Space of Possible Program Optimizations

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]
```

Search Space Example (3/3)



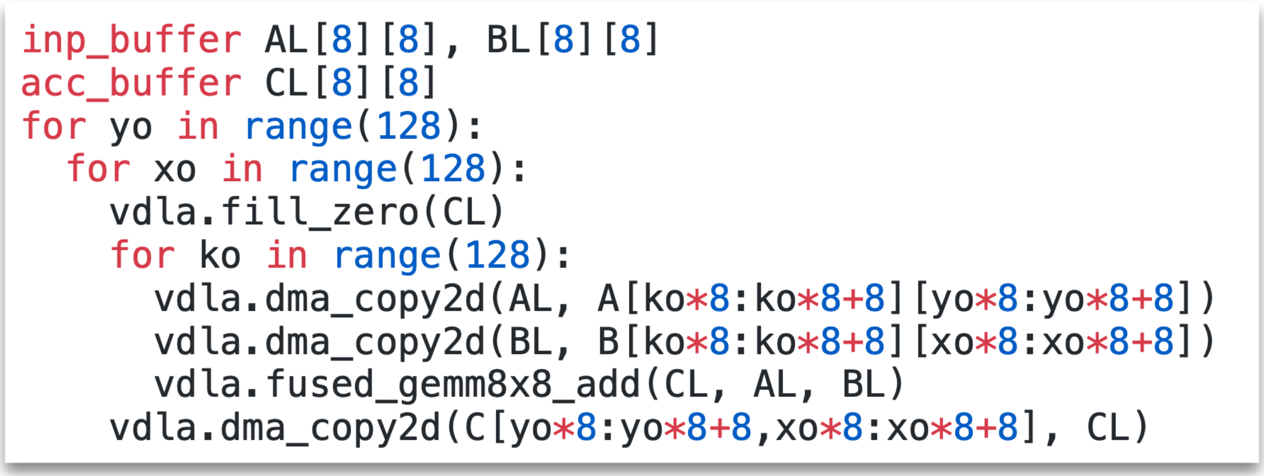
Tensor Expression (Specification)

C = tvm.compute((m, n),

lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))

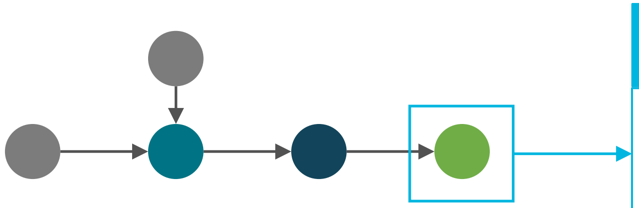
Search Space of Possible Program Optimizations

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

Optimization space is really large…



Tensor Expression (Specification)

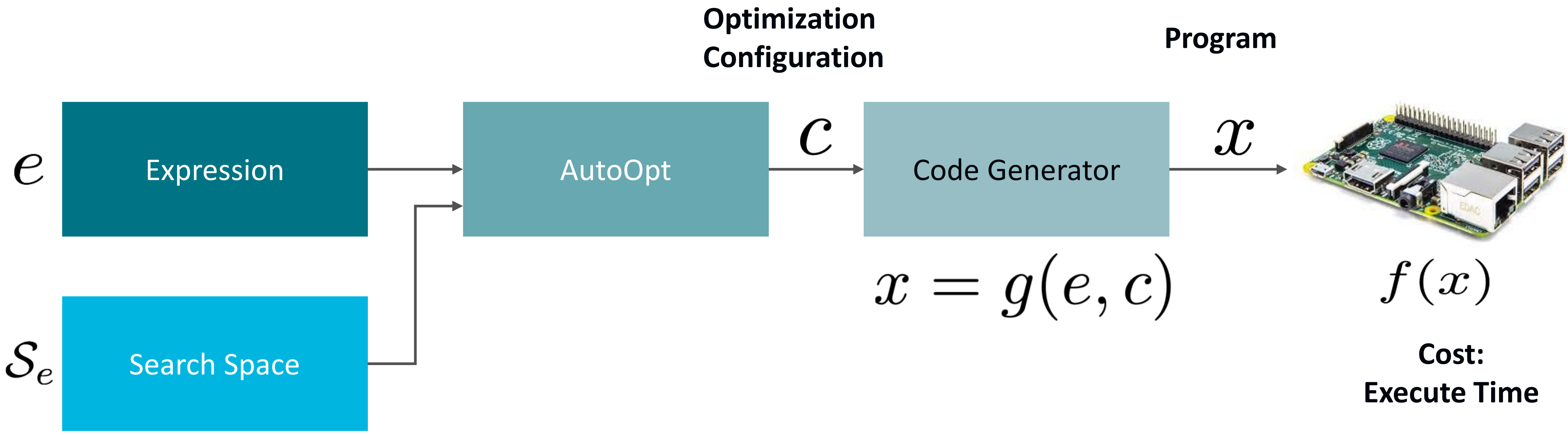
```
C = tvm.compute((m, n),
lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Billions of possible optimization choices

Loop Transformations	Thread Bindings	Cache Locality
Thread Cooperation	Tensorization	Latency Hiding

Typically explored via human intuition.
How can we automate this? Auto-tuning is too slow.

Problem Formalization



e Expression

S_e Search Space

AutoOpt

Optimization Configuration

c

Code Generator

$x = g(e, c)$

Program

x

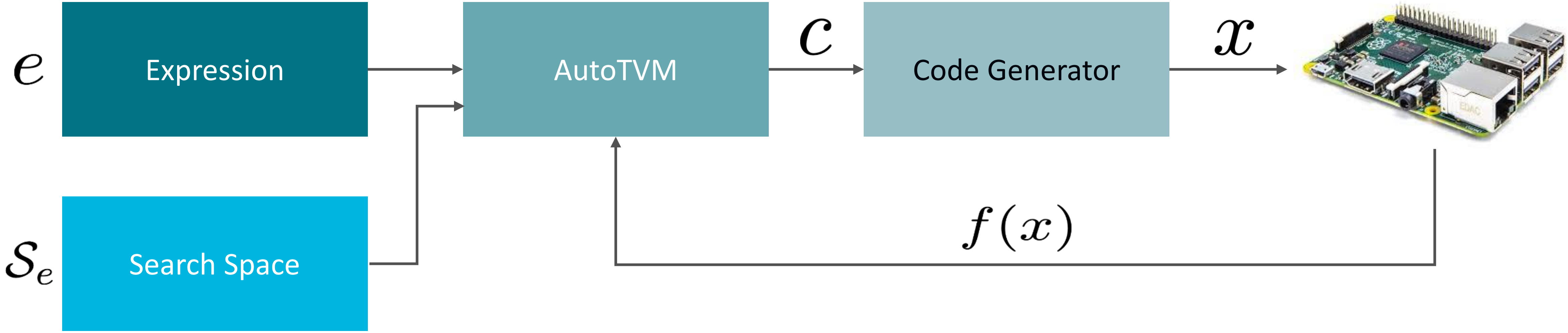
$f(x)$

Cost: Execute Time

Objective $$argmin_{c \in S_e} f(g(e,c))$$

Black-box Optimization

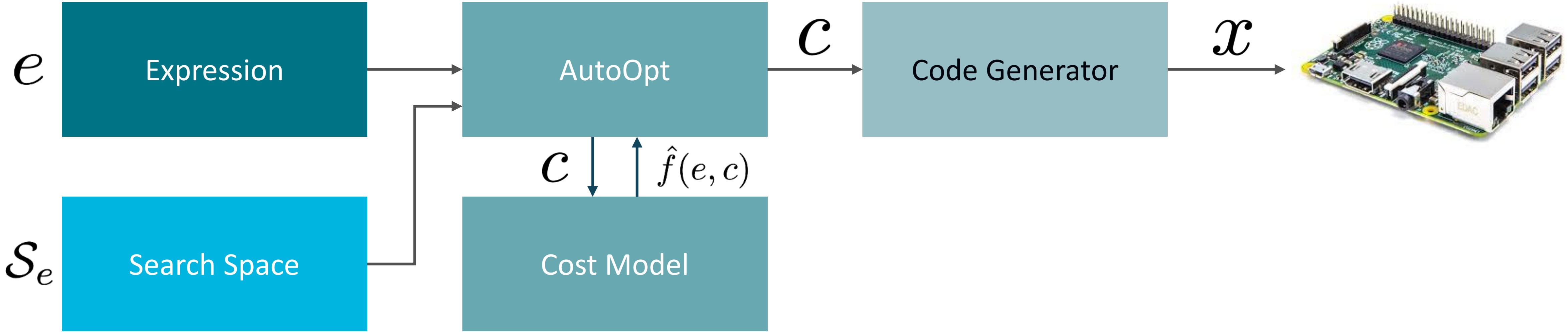
Try each configuration c until we find a good one



Challenge: Lots of experimental trials, each trial costs ~1 second

Cost-model Driven Approach

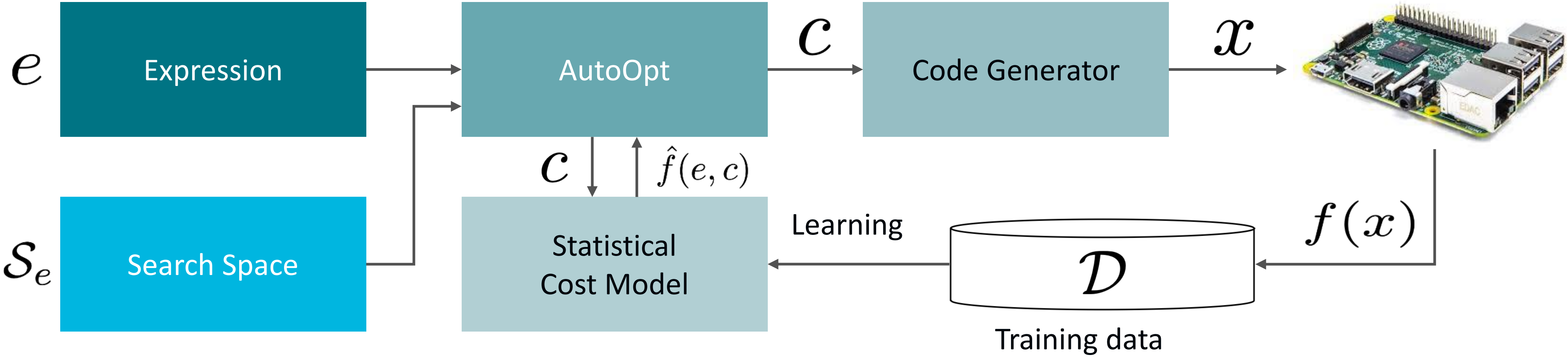
Use cost model to pick configuration



Challenge: Need reliable cost model per hardware

Statistical Cost Model

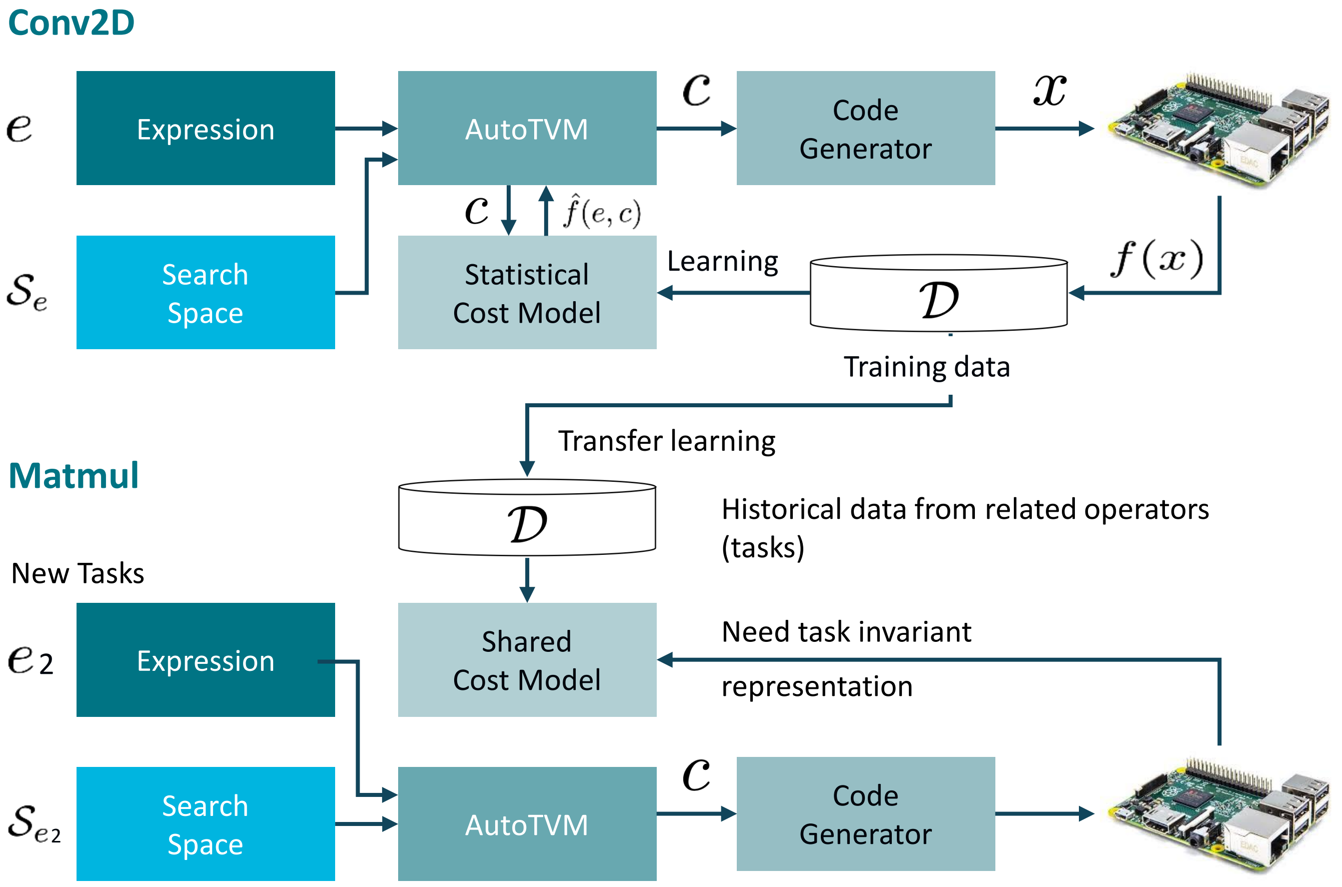
Our approach: Use machine learning to learn a statistical cost model



Benefit: Automatically adapt to hardware type

Important: How to design the cost model

AutoTVM Overview



Conv2D

e Expression → AutoTVM

S_e Search Space → AutoTVM

AutoTVM $\xrightarrow{c}$ Code Generator $\xrightarrow{x}$

AutoTVM c ↓ ↑ $\hat{f}(e,c)$ Statistical Cost Model

$f(x)$ → $\mathcal{D}$ Training data

Learning

Transfer learning

Matmul

$\mathcal{D}$ Historical data from related operators (tasks)

New Tasks

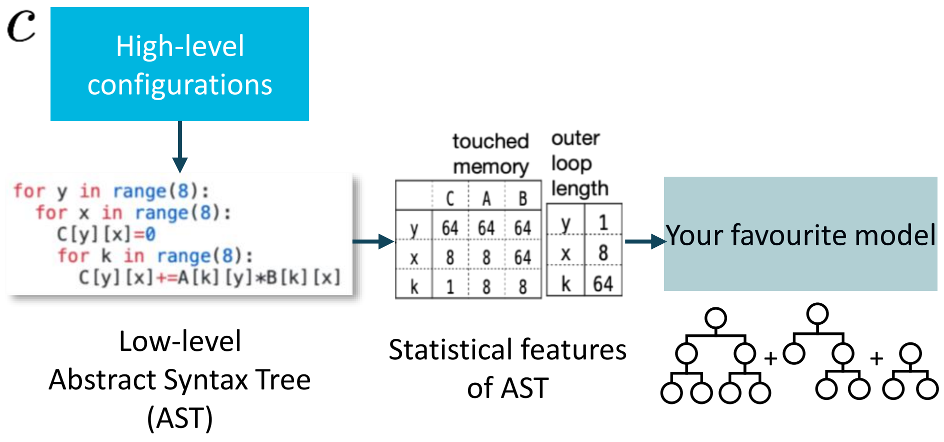
e_2 Expression

S_{e2} Search Space

Shared Cost Model

Need task invariant representation

AutoTVM $\xrightarrow{c}$ Code Generator



c High-level configurations

```
for y in range(8):
  for x in range(8):
    C[y][x]=0
    for k in range(8):
      C[y][x]+=A[k][y]*B[k][x]
```

Low-level Abstract Syntax Tree (AST)

touched memory

	C	A	B
y	64	64	64
x	8	8	64
k	1	8	8

outer loop length

y	1
x	8
k	64

Statistical features of AST

Your favourite model

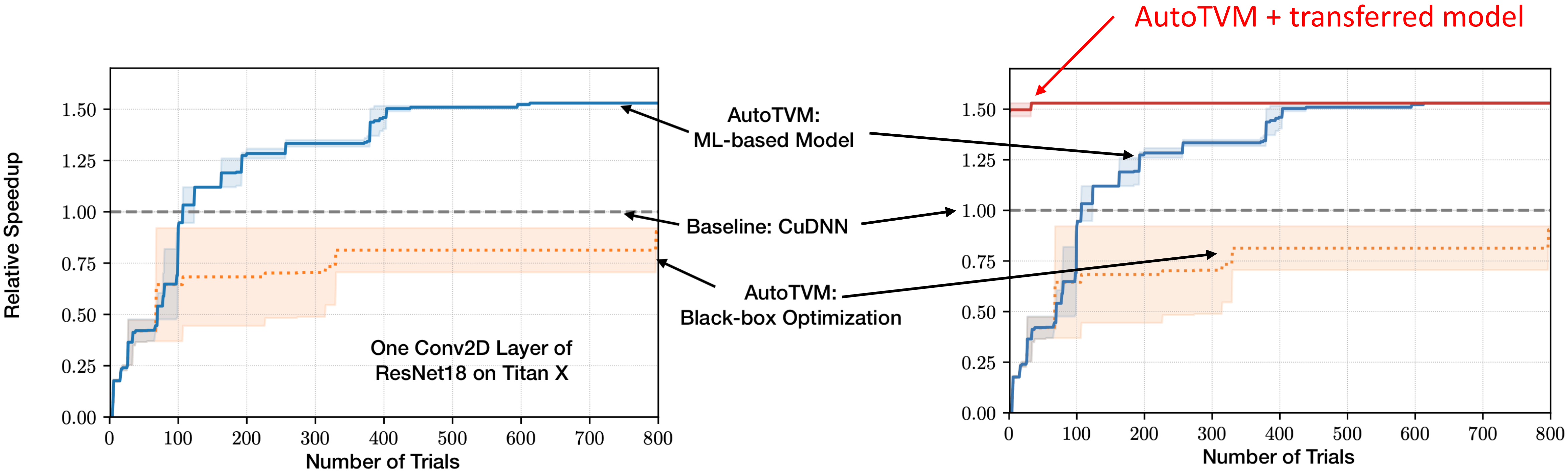
Benefit: Low-level AST is a common representation (General, task invariant)

	Task Invariant	Time Cost	Predictive Accuracy
Vanilla Model	No	Low	Medium
Tree-based Model	Yes	Low	Good
Neural Model	Yes	High	Good

O(microseconds) inference vs. O(seconds) execution

Learning to Optimize Tensor Programs. Chen et al. NeurIPS 18

Does it work?



AutoTVM + transferred model

AutoTVM: ML-based Model

Baseline: CuDNN

AutoTVM: Black-box Optimization

One Conv2D Layer of ResNet18 on Titan X

Better than hand-tuned code in a few minutes

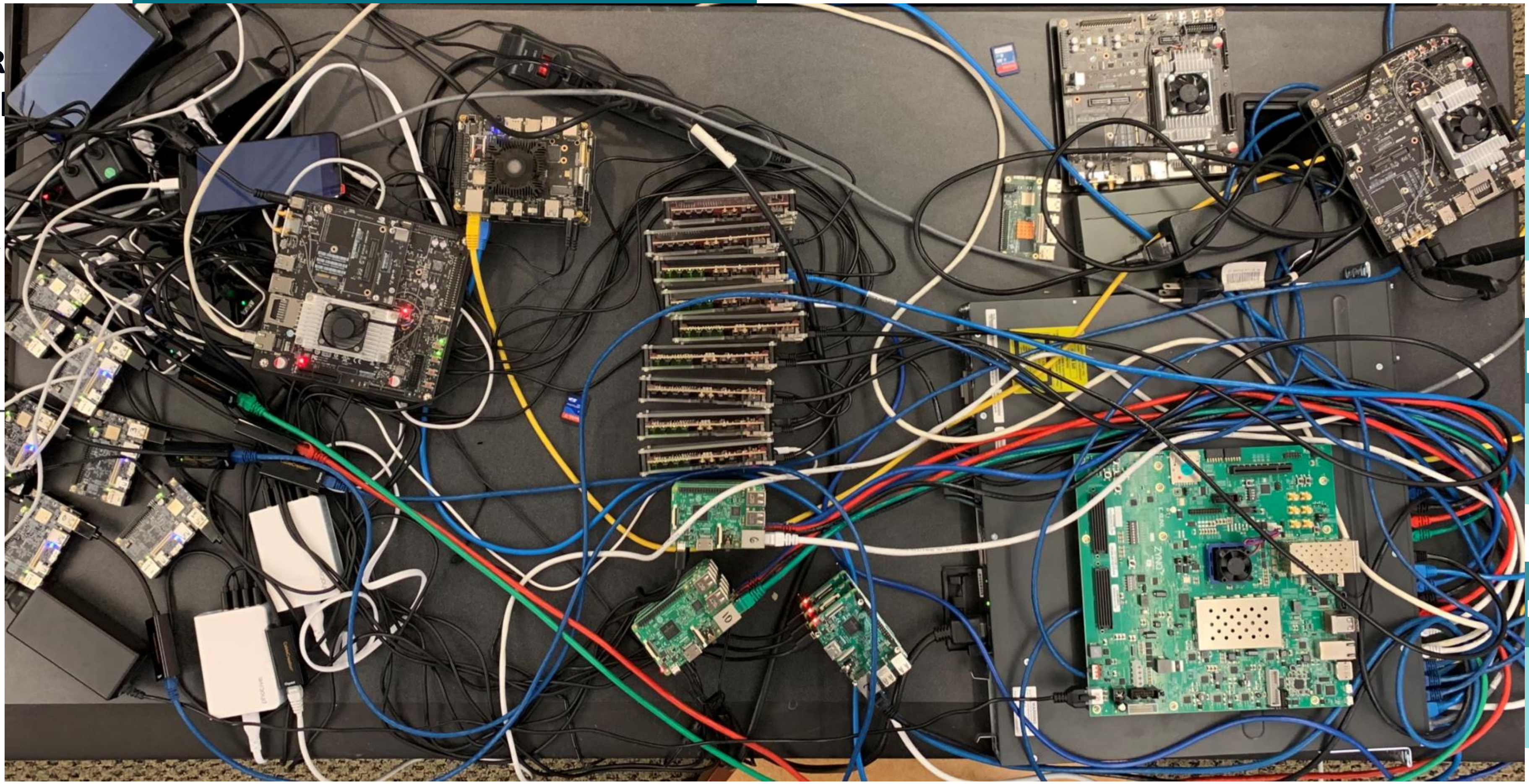
1.50x faster than hand-tuned in steady state

3x to 10x faster tuning w/ transfer learning

Device Fleet: Distributed Test Bed for AutoTVM

Resource Manager (Tracker)

R
Al



AutoTVM
Experiment 1

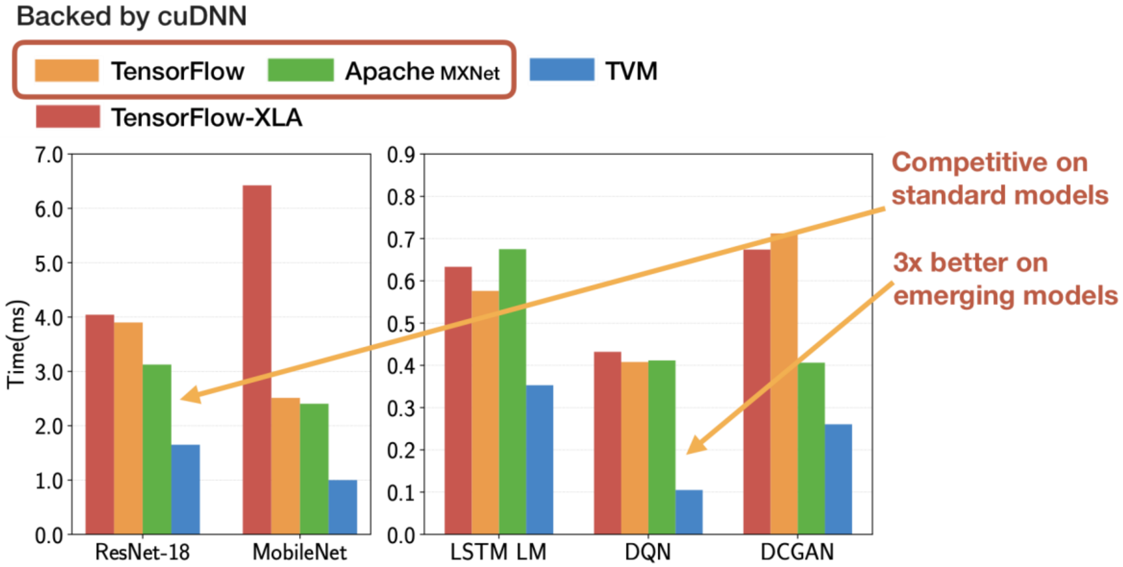
AutoTVM
Experiment 2

L

m

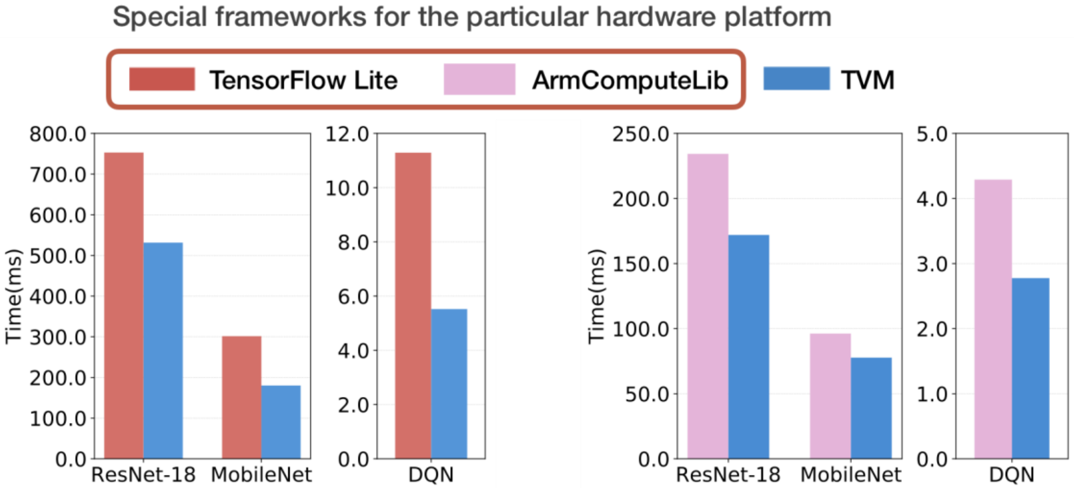
...

State-of-the-art performance



Competitive on standard models

3x better on emerging models



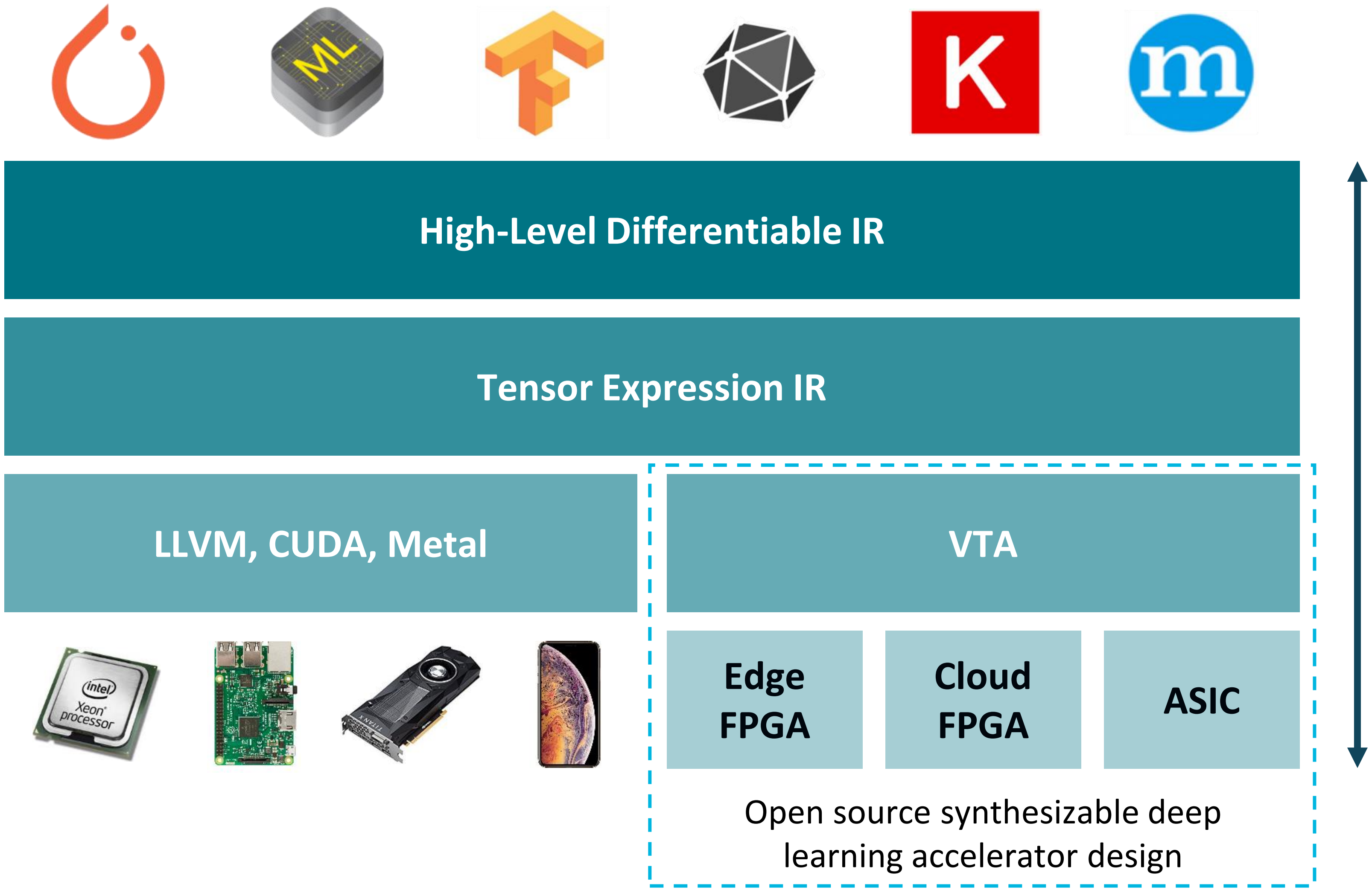
Nvidia Titan X

ARM CPU (Cortex-A53)

ARM GPU (MALI)

Key point: TVM offers good performance with low manual effort

tvm Stack



High-Level Differentiable IR

Tensor Expression IR

LLVM, CUDA, Metal

VTA

Edge FPGA

Cloud FPGA

ASIC

Open source synthesizable deep learning accelerator design

End-to-end, framework to metal open stack. Research and deployment

DL Accelerator Design Challenges

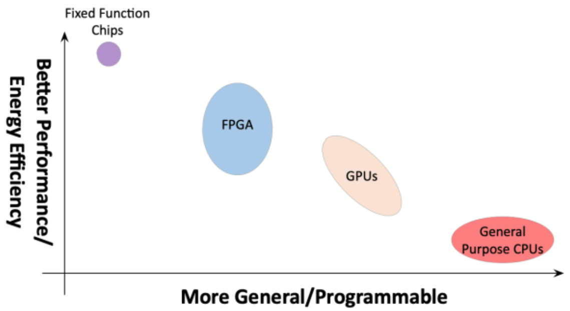
- Keeping up with algorithmic changes
  - (VTA: two-level ISA, templatized design)

GAN MLP

RNN

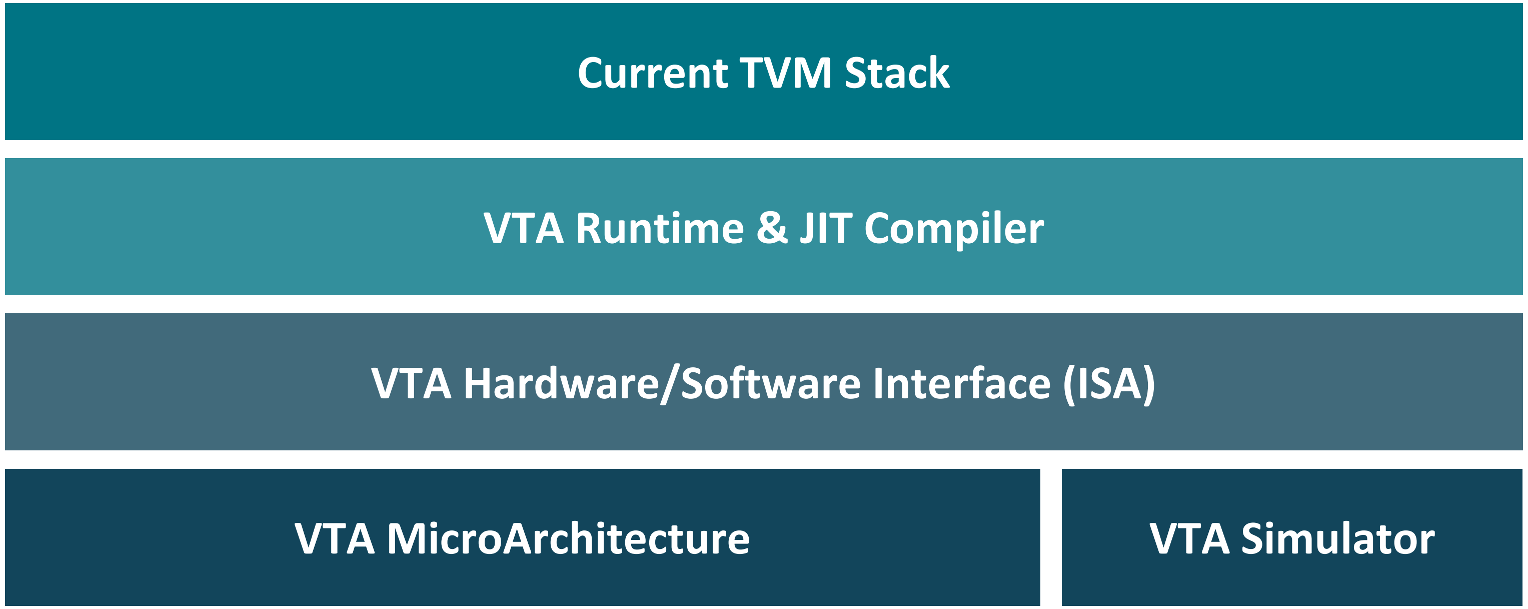
CNN DQNN

- Finding the right generality/efficiency trade-off
  - (VTA: templatized design + HW parameter search)



- Enable a “day-0” software stack on top
  - (VTA: tight coupling with TVM)

VTA: Open & Flexible Deep Learning Accelerator

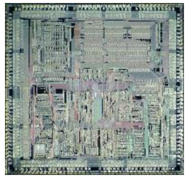

Current TVM Stack

VTA Runtime & JIT Compiler

VTA Hardware/Software Interface (ISA)

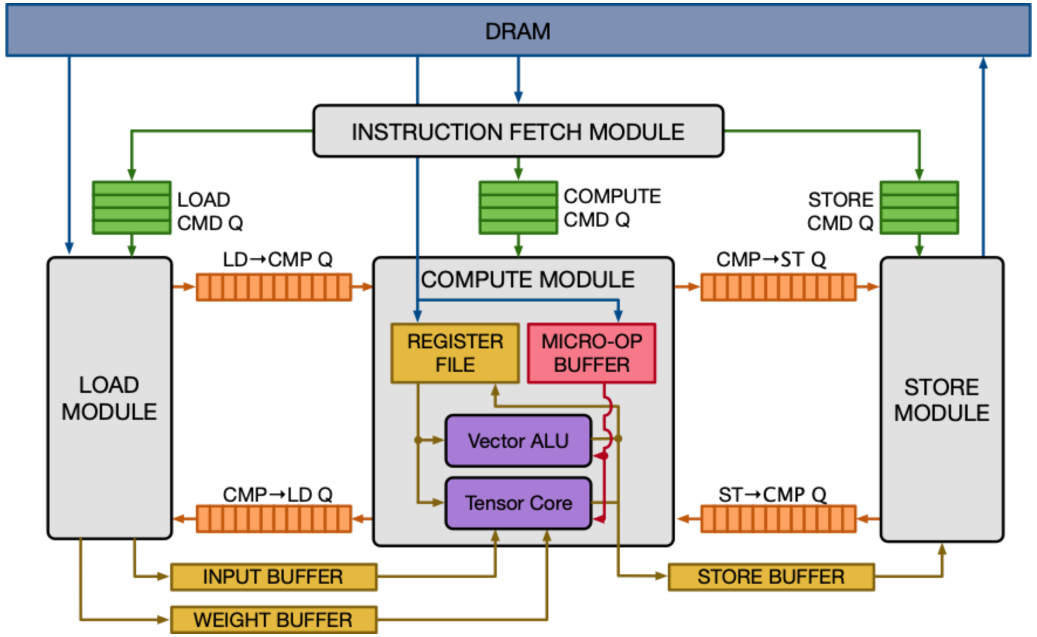
VTA MicroArchitecture

VTA Simulator

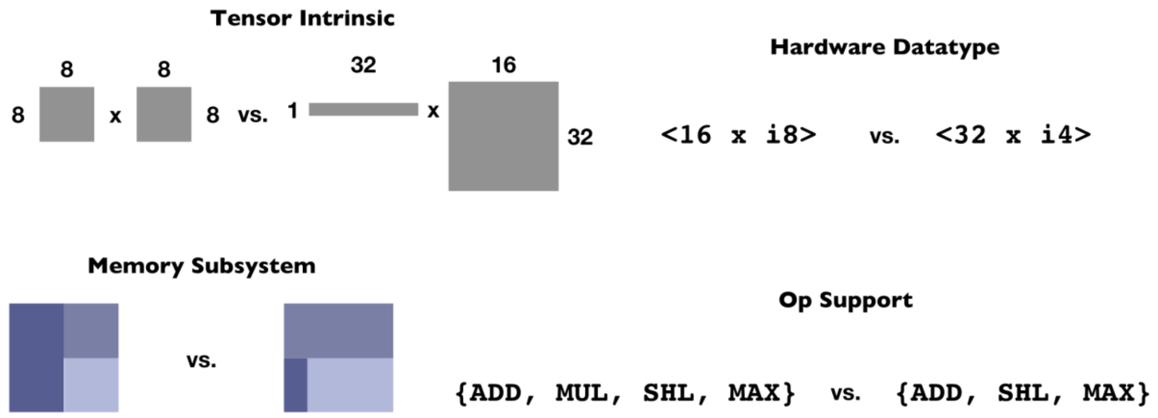


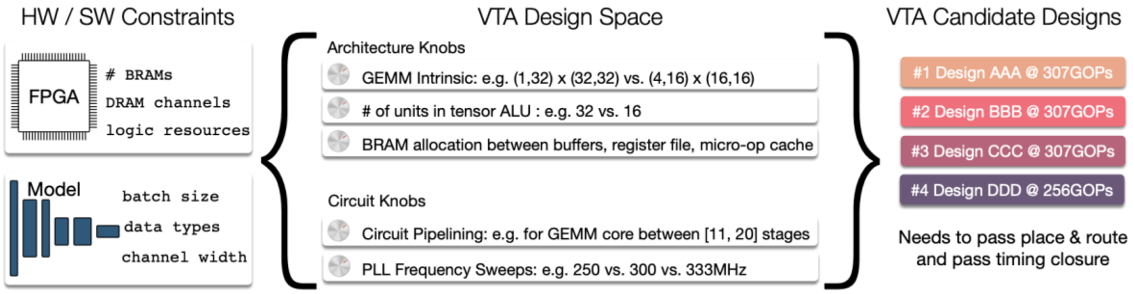
- Move hardware complexity to software via a **two-level ISA**
- Runtime **JIT-compile accelerator micro code**
- Native support in TVM
- Support heterogenous devices (split graph)
- Support for secure execution (soon)

VTA Open Source Deep Learning accelerator



Template



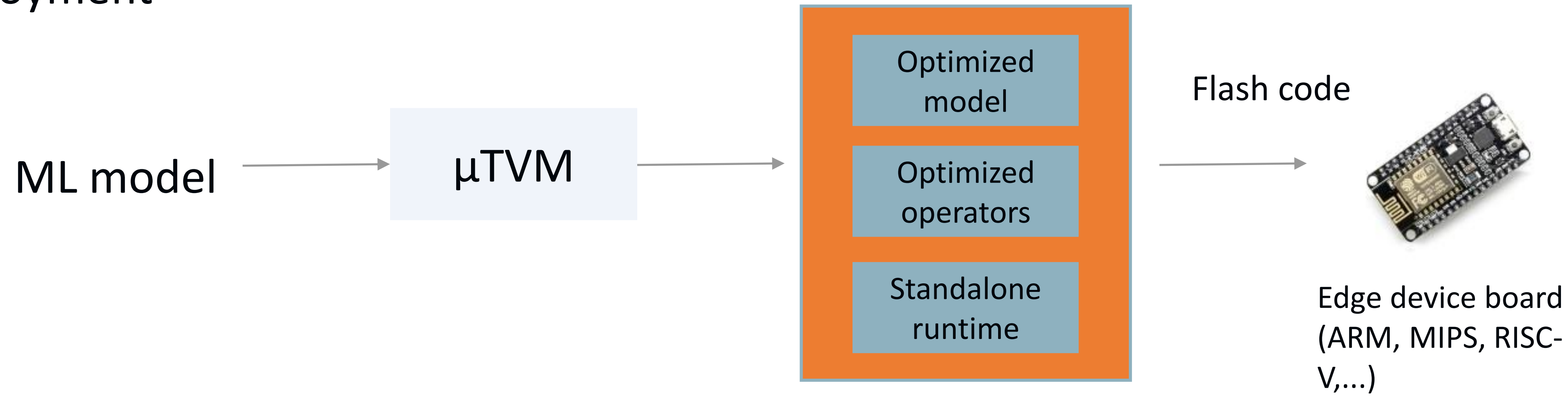

HW / SW Constraints
FPGA
# BRAMs
DRAM channels
logic resources
Model
batch size
data types
channel width
VTA Design Space
Architecture Knobs
GEMM Intrinsic: e.g. (1,32) x (32,32) vs. (4,16) x (16,16)
# of units in tensor ALU : e.g. 32 vs. 16
BRAM allocation between buffers, register file, micro-op cache
Circuit Knobs
Circuit Pipelining: e.g. for GEMM core between [11, 20] stages
PLL Frequency Sweeps: e.g. 250 vs. 300 vs. 333MHz
VTA Candidate Designs
#1 Design AAA @ 307GOPs
#2 Design BBB @ 307GOPs
#3 Design CCC @ 307GOPs
#4 Design DDD @ 256GOPs
Needs to pass place & route and pass timing closure

- Decoupled access-execute with explicit software control
- Two-level ISA: JIT breaks multi-cycle “CISC” instructions into micro-ops
  - Enables model retargeting without HW changes
- Focused on FPGA deployments so far. Exploring custom silicon possibilities

Note: HW-SW Blueprint for Flexible Deep Learning Acceleration. Moreau et al. IEEE Micro 2019.

µTVM - Bare-metal model deployment for edge devices

Optimize, compile and package model for standalone bare metal deployment



ML model

µTVM

Optimized model

Optimized operators

Standalone runtime

Flash code

Edge device board (ARM, MIPS, RISC-V,...)

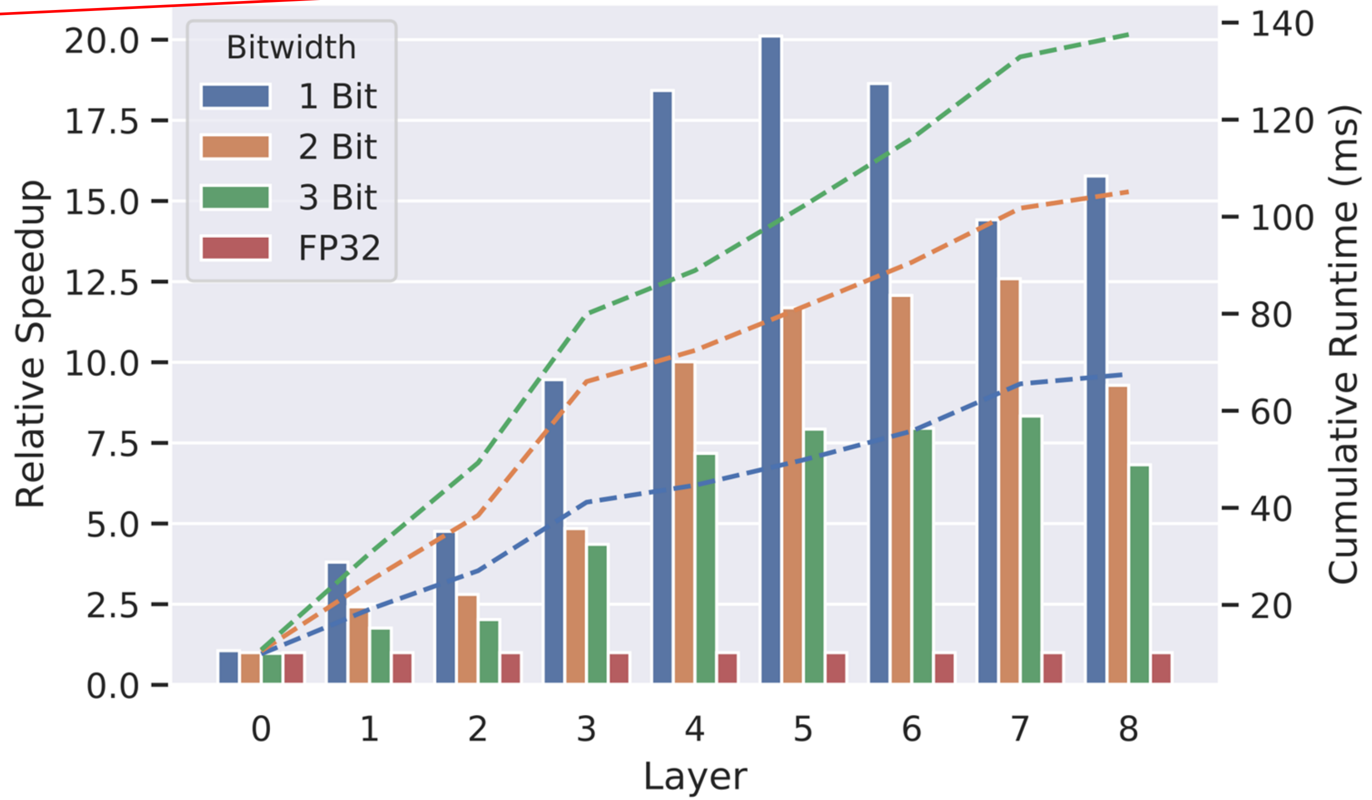
See recent demo on TVM for Azure Sphere deployment.

Coming Soon - Ultra low bit-width quantization

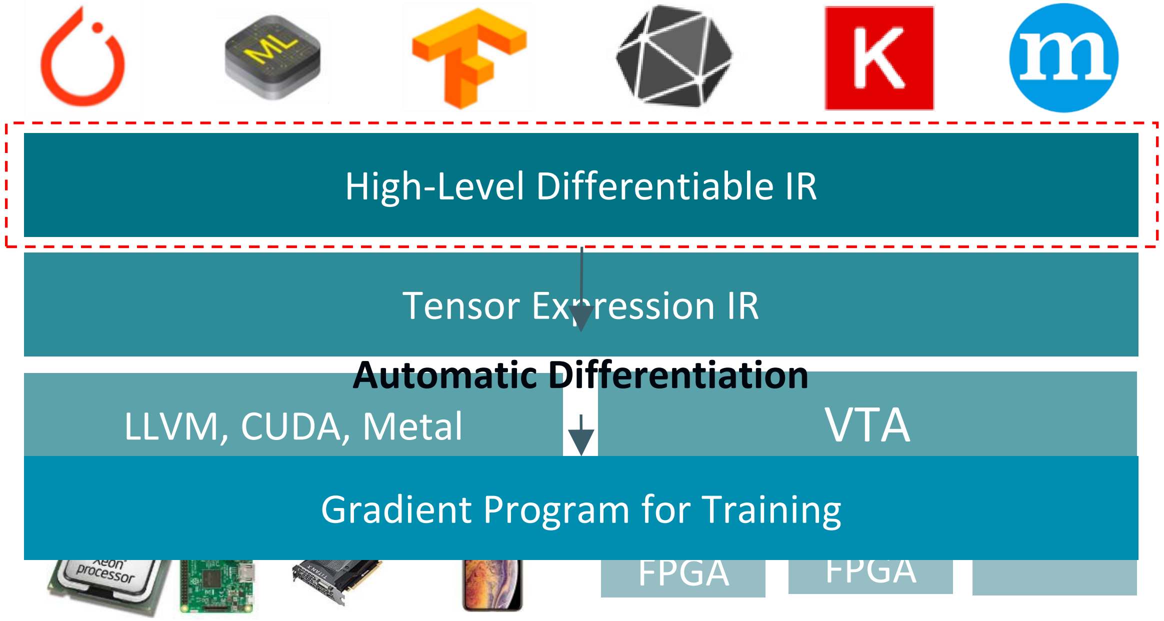
Automatic quantization: 5-20x performance gains with reasonable accuracy loss.

TVM supports flexible code generation for a variety of data types

Squeezenet on RaspberryPi 3



What about training?



High-Level Differentiable IR

Tensor Expression IR

Automatic Differentiation

LLVM, CUDA, Metal

VTA

Gradient Program for Training

FPGA

FPGA

Standalone inference deployment

Standalone training deployment

- Direct support for training in Apache TVM coming soon!
- Automatic generation of gradient programs
- Support for customized data types and training on FPGAs

Other Ongoing TVM efforts

- Autoscheduling (Zheng et al. OSDI’20 @ UCBerkeley)
- Automatic synthesis of operator implementations (Cowan et al. CGO’20 @ UWash)
- Sparse support (NLP, graph convolutional neural networks, etc…)
- Secure enclaves
- …
- Join the community!

https://tvm.ai

2nd TVM conference on Dec 5, 2019. 200+ ppl last year!

```
import tvm
from tvm import rpc, autotvm
from tvm.contrib import graph_runtime, util
from tvm.contrib.download import download
import nnvm.compiler
import vta
import vta.testing
```

3rd TVM conference on Dec 3/4, 2020. https://tvmconf.org

- Video tutorials
- iPython notebooks tutorials

Drive TVM adoption

Core infrastructure and improvements

Product: SaaS automation for ML ops

Optimizing, benchmarking, and packaging models for deployment

Support

TVM end users and hardware vendors

Apache TVM ecosystem
OctoML

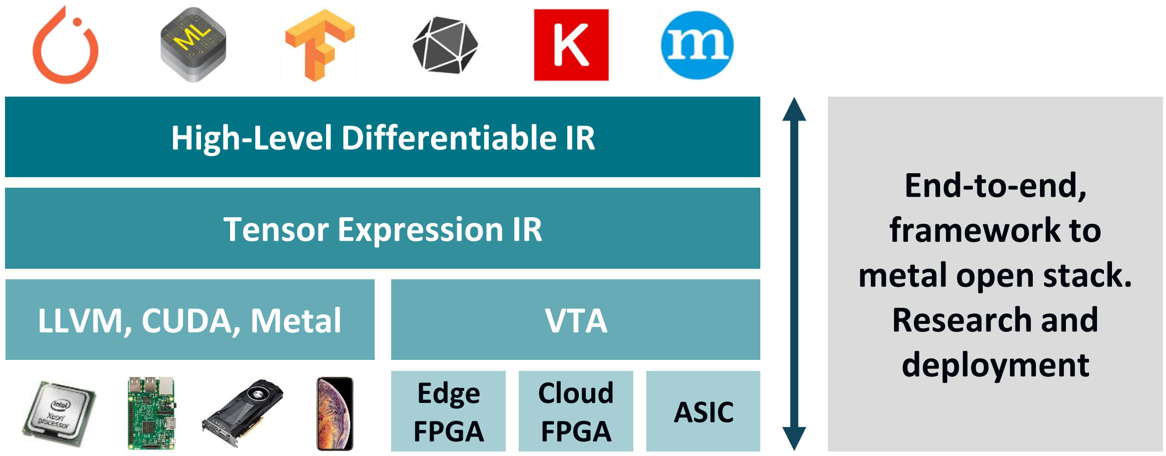
https://octoml.ai

What I would like you to remember…

TVM is an emerging **open source** standard for ML compilation and optimization

TVM offers

- Improved time to market for ML
- Performance
- Unified support for CPU, GPU, Accelerators
- On the framework of your choice


ML
K
m
High-Level Differentiable IR
Tensor Expression IR
LLVM, CUDA, Metal
VTA
Edge FPGA
Cloud FPGA
ASIC
End-to-end, framework to metal open stack. Research and deployment

OctoML is here to help you succeed in you ML deployment needs

tvm.ai

OctoML

W PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING