tinyML® Talks

Enabling Ultra-low Power Machine Learning at the Edge

"An Introduction to Optimizing ML Models with TVMC"

Chris Hoge - OctoML

Seattle Area Group – February 18, 2021

TINY ML

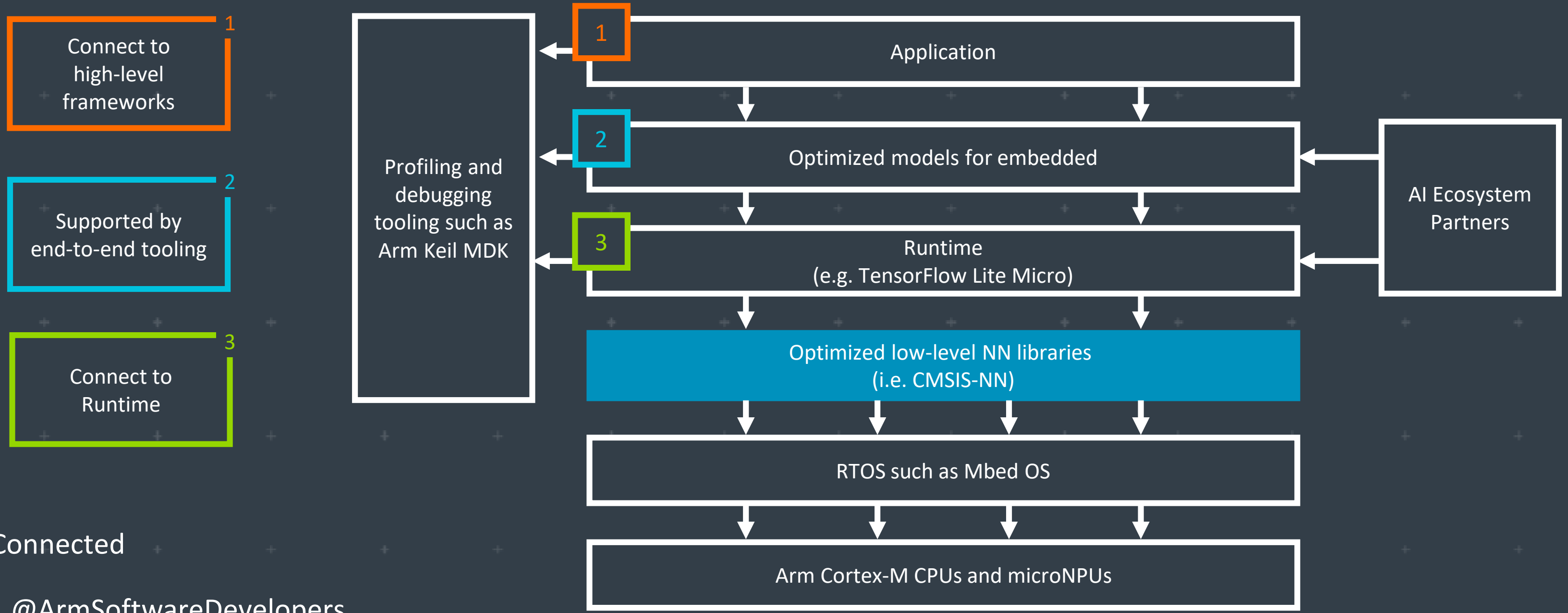
www.tinyML.org

tinyML Talks Sponsors

tinyML Strategic Partner

Additional Sponsorships available – contact sponsorships@tinyML.org for info

Arm: The Software and Hardware Foundation for tinyML



Connect to high-level frameworks 1
Supported by end-to-end tooling 2
Connect to Runtime 3
Profiling and debugging tooling such as Arm Keil MDK
1
Application
2
Optimized models for embedded
3
Runtime (e.g. TensorFlow Lite Micro)
Optimized low-level NN libraries (i.e. CMSIS-NN)
RTOS such as Mbed OS
Arm Cortex-M CPUs and microNPUs
AI Ecosystem Partners

Stay Connected

@ArmSoftwareDevelopers

@ArmSoftwareDev

Resources: developer.arm.com/solutions/machine-learning-on-arm

© 2020 Arm Limited (or its affiliates)

Deeplite

WE USE AI TO MAKE OTHER AI FASTER, SMALLER AND MORE POWER EFFICIENT

Automatically compress SOTA models like MobileNet to <200KB with **little to no drop in accuracy** for inference on resource-limited MCUs

Reduce model optimization trial & error from weeks to days using Deeplite's **design space exploration**

APPLY NOW

Deploy more models to your device without sacrificing performance or battery life with our **easy-to-use software**

BECOME BETA USER bit.ly/testdeeplite

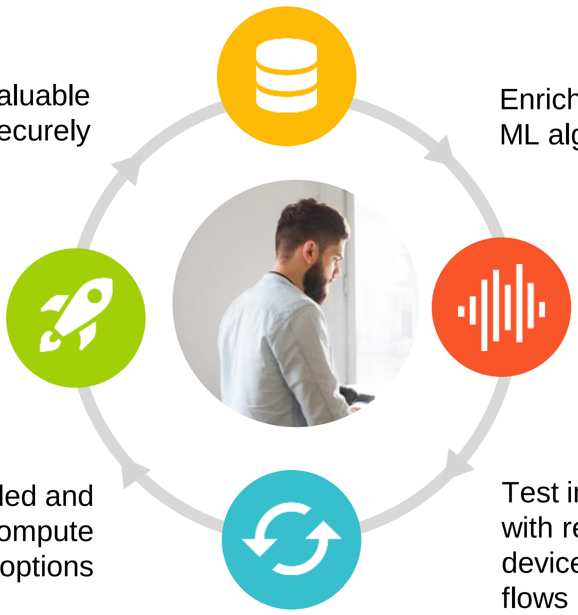
mobilityXlab arm AI 100 CBINSIGHTS

TinyML for all developers

C++ library

Arduino library

WebAssembly



Dataset

Acquire valuable training data securely

Enrich data and train ML algorithms

Impulse

Edge Device

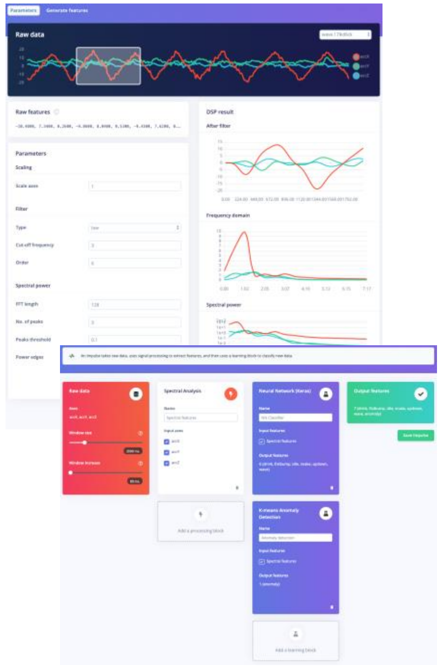
Real sensors in real time

Open source SDK

Embedded and edge compute deployment options

Test impulse with real-time device data flows

Test



Copyright © EdgeImpulse Inc.

Get your free account at http://edgeimpulse.com

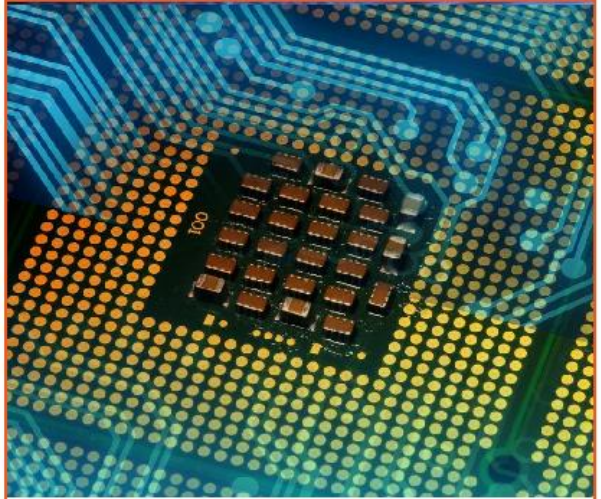
Maxim Integrated: Enabling Edge Intelligence

www.maximintegrated.com/ai

Sensors and Signal Conditioning

Health sensors measure PPG and ECG signals critical to understanding vital signs. Signal chain products enable measuring even the most sensitive signals.

Low Power Cortex M4 Micros



The biggest (3MB flash and 1MB SRAM) and the smallest (256KB flash and 96KB SRAM) Cortex M4 microcontrollers enable algorithms and neural networks to run at wearable power levels

Advanced AI Acceleration

The new MAX78000 implements AI inferences at over 100x lower energy than other embedded options. Now the edge can see and hear like never before.

Qeexo AutoML for Embedded AI

Automated Machine Learning Platform that builds tinyML solutions for the Edge using sensor data

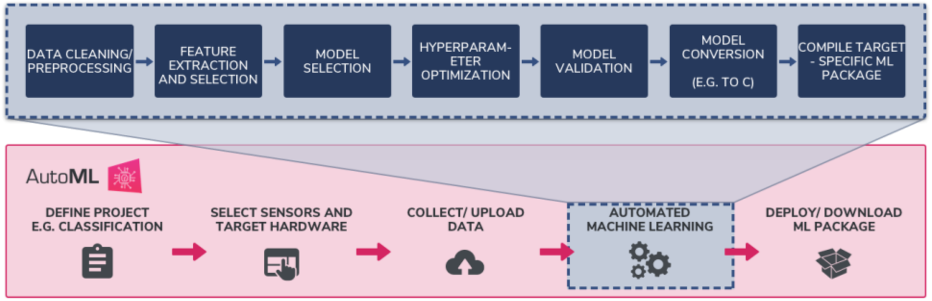
Key Features

- Wide range of ML methods: GBM, XGBoost, Random Forest, Logistic Regression, Decision Tree, SVM, CNN, RNN, CRNN, ANN, Local Outlier Factor, and Isolation Forest
- Easy-to-use interface for labeling, recording, validating, and visualizing time-series sensor data
- On-device inference optimized for low latency, low power consumption, and a small memory footprint
- Supports Arm® Cortex™- M0 to M4 class MCUs
- Automates complex and labor-intensive processes of a typical ML workflow – no coding or ML expertise required!

Target Markets/Applications

- Industrial Predictive Maintenance
- Smart Home
- Wearables
- Automotive
- Mobile
- IoT

QEEXO AUTOML: END-TO-END MACHINE LEARNING PLATFORM


DATA CLEANING/ PREPROCESSING
FEATURE EXTRACTION AND SELECTION
MODEL SELECTION
HYPERPARAM-ETER OPTIMIZATION
MODEL VALIDATION
MODEL CONVERSION (E.G. TO C)
COMPILE TARGET - SPECIFIC ML PACKAGE
AutoML
DEFINE PROJECT E.G. CLASSIFICATION
SELECT SENSORS AND TARGET HARDWARE
COLLECT/ UPLOAD DATA
AUTOMATED MACHINE LEARNING
DEPLOY/ DOWNLOAD ML PACKAGE

For a limited time, sign up to use Qeexo AutoML at automl.qeexo.com for FREE to bring intelligence to your devices!

Reality AI®

is for
building products

https://reality.ai info@reality.ai @SensorAI Reality AI

Reality AI Tools® software

Automated Feature Exploration and Model Generation

Bill-of-Materials Optimization

Automated Data Assessment

Edge AI / TinyML code for the smallest MCUs

Reality AI solutions

Automotive sound recognition & localization

Indoor/outdoor sound event recognition

RealityCheck™ voice anti-spoofing

SynSense

SynSense builds **ultra-low-power** (sub-mW) **sensing and inference** hardware for **embedded, mobile and edge** devices. We design systems for **real-time always-on smart sensing**, for audio, vision, IMUs, bio-signals and more.

https://SynSense.ai

Next tinyML Talks

Date	Presenter	Topic / Title
Tuesday, March 2	**Eben Upton** founder of the Raspberry Pi Foundation	Inference with Raspberry Pi Pico and RP2040

Webcast start time is 8 am Pacific time

Please contact talks@tinyml.org if you are interested in presenting

Local Committee in Seattle

Karl Fezer

Contact: karl@tinyml.org

Announcement

TINY ML
tinyML Summit 2021
Enabling ultra-low Power Machine Learning at the Edge
March 22-26, 2021 | Online

www.tinyML.org/summit2021

Highlights:

- Keywords: Premier Quality, Interactive, LIVE … and FREE
- 5 days, 50+ presentations
- 4 Tutorials
- 2 Panel discussions: (i) VC and (ii) tinyML toolchains
- tinyML Research Symposium
- Late Breaking News
- 3 Best tinyML Awards (Paper, Product, Innovation)
- 10+ Breakout sessions on various topics
- tinyML Partner sessions
- tinyAI for (Good) Life
- LIVE coverage, starting at 8am Pacific time

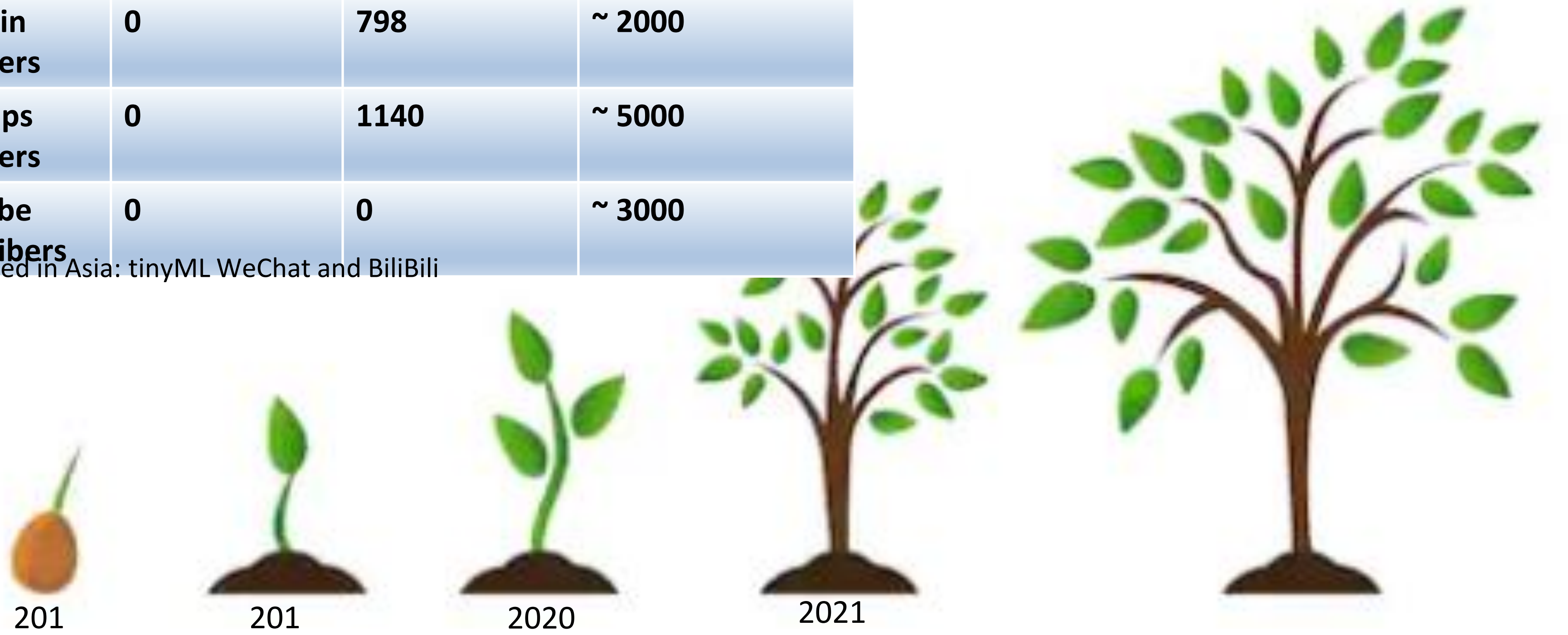
What should I do about it:

- Check out the program – you will be impressed
- Register on-line (takes 5 min)
- If interested: Submit nominations for Best Awards and/or Late News – February 28 deadline
- Block out your calendar: March 22-26
- Become a sponsor (sponsorships@tinyML.org)
- Actively participate at the Summit
- Provide your feedback – we listen !
- Don't worry about missing some talks – all videos will be posted on YouTube.com/tinyML

tinyML is growing fast

	2019 Summit *(March 2019)*	2020 Summit *(Feb 2020)*	2021 Summit *(March 2021), expected*
Attendees	**160**	**400+**	**3000+**
Companies	**90**	**172**	**300+ (?)**
Linkedin members	**0**	**798**	**~ 2000**
Meetups members	**0**	**1140**	**~ 5000**
YouTube subscribers	**0**	**0**	**~ 3000**

also started in Asia: tinyML WeChat and BiliBili



Summit Sponsors

(as of Feb 15, 2021)

Contact: sponsorships@tinyML.org

multiple levels and benefits available
(also check www.tinyML.org)

Executive Sponsors

arm Qualcomm
SAMSUNG

Platinum Sponsors

Eta Compute DISRUPTION AT THE EDGE
LATTICE SEMICONDUCTOR

Gold Sponsors

brainchip DSP GROUP EDGE IMPULSE CISCO Google
emza visual sense GML GrAI Matter Labs GREENWAVES TECHNOLOGIES SAP
imagimob Qeexo RealityAI SensiML
Himax SILICON LABS XMOS SYNTIANT LatentAI

Silver Sponsors

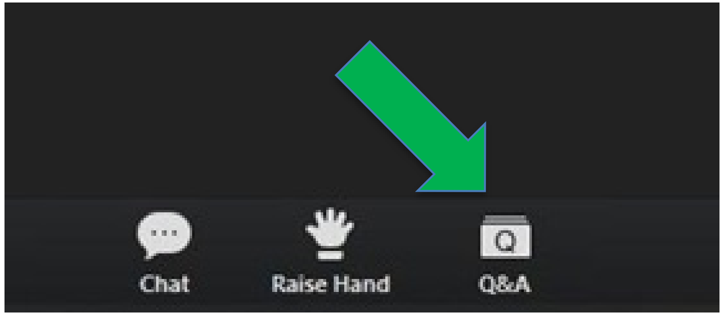
EDGECORTIX HOTG SynSense

Reminders

Slides & Videos will be posted tomorrow

tinyml.org/forums youtube.com/tinyml

Please use the Q&A window for your questions



TINY ML TALKS webcast

Chris Hoge

Chris has more than 10 years of experience in helping build open source communities, including strategic program management for the OpenStack Foundation and as a SIG co-founder in the Kubernetes community. He has a background in Applied Mathematics, and lives in the PNW.

OctoML

An Introduction to Apache TVM and TVMC

Chris Hoge, Developer Advocate for Apache TVM

choge@octoml.ai

Why should you use TVM?

How does TVM work?

How do you install TVM?

How do you use TVMC?

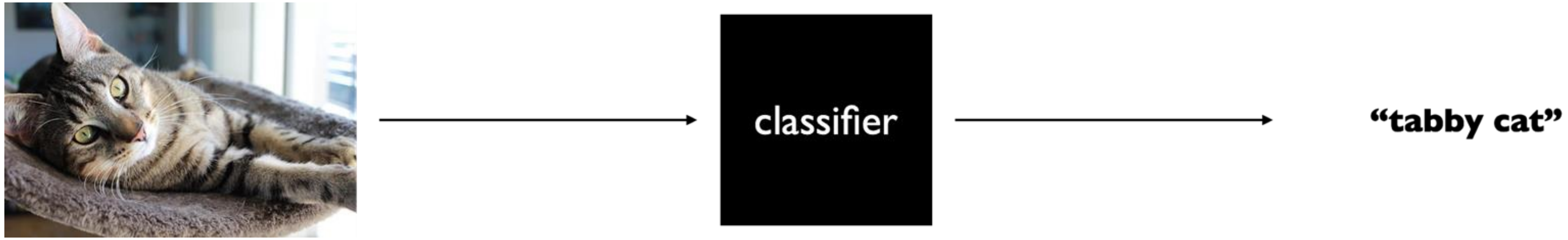
Why should you use TVM?

How does TVM work?

How do you install TVM?

How do you use TVMC?

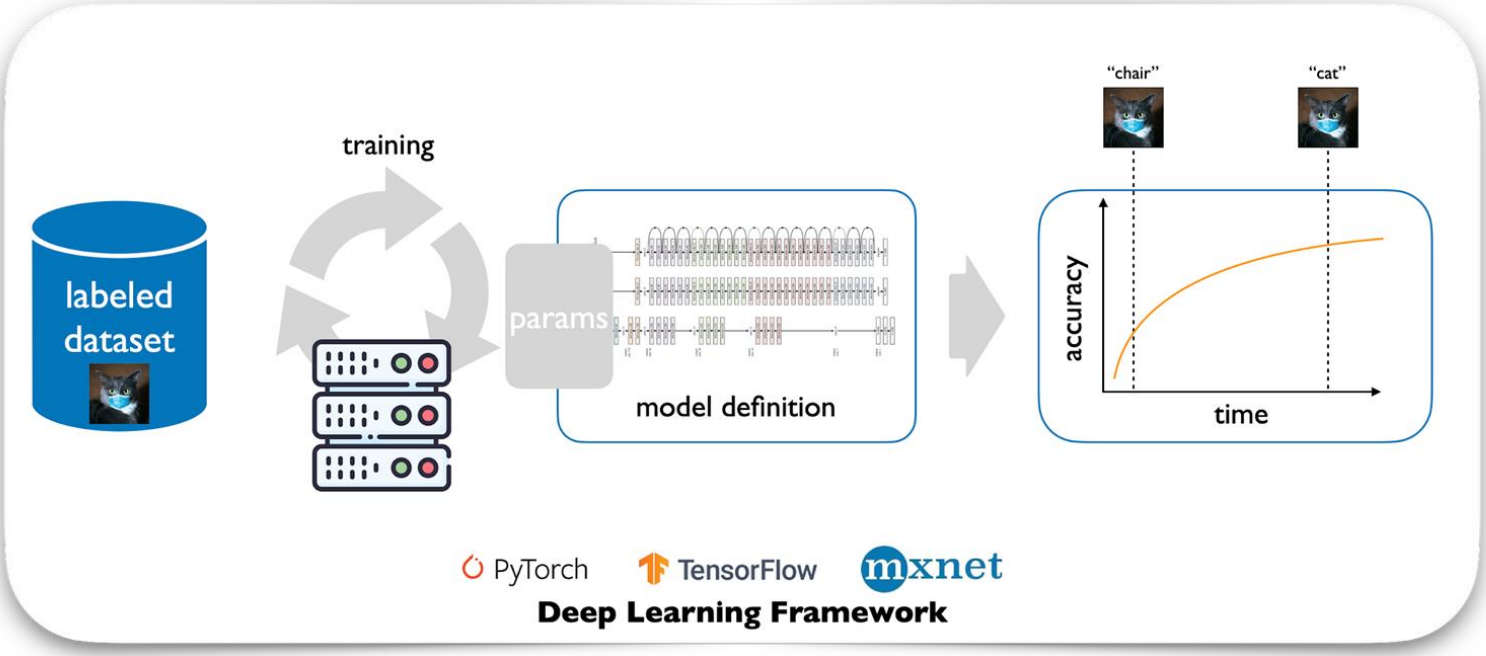
A Simple Image Classification Example



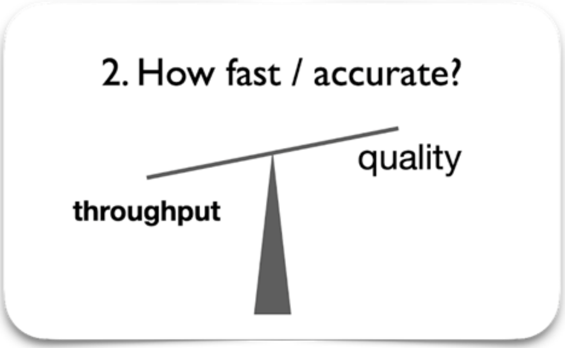
Phase I: Train the Classifier (training)

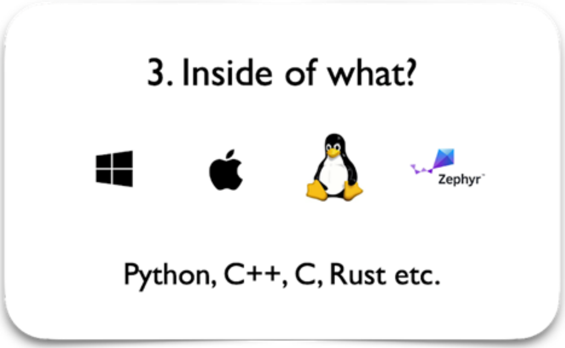
Phase II: Deploy the Classifier (inference)

Phase I: Train the Classifier (we are not concerned with this…)



Phase 2: Deploy the Classifier (what we are interested in with this talk...)





Let the Deployment Challenge Begin!

PyTorch	?	?	?	?	?	?	?	?	?	?
Caffe2	?	?	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
ONNX	?	?	?	?	?	?	?	?	?	?
TensorFlow Lite	?	?	?	?	?	?	?	?	?	?

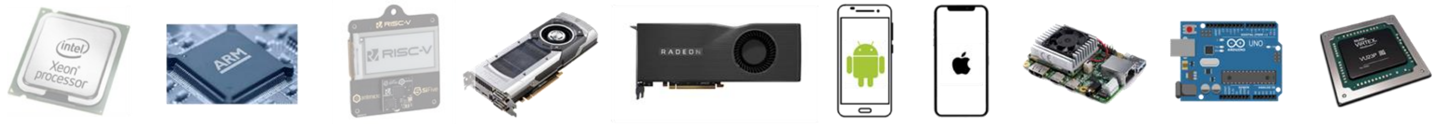
OctoML

Let the Deployment Challenge Begin!

PyTorch	?	?	?	?	?	?	?	?	?	?
Caffe2	?	?	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
ONNX	?	?	?	?	?	?	?	?	?	?
TensorFlow Lite	?	✓	?	?	?	?	?	?	?	?

OctoML

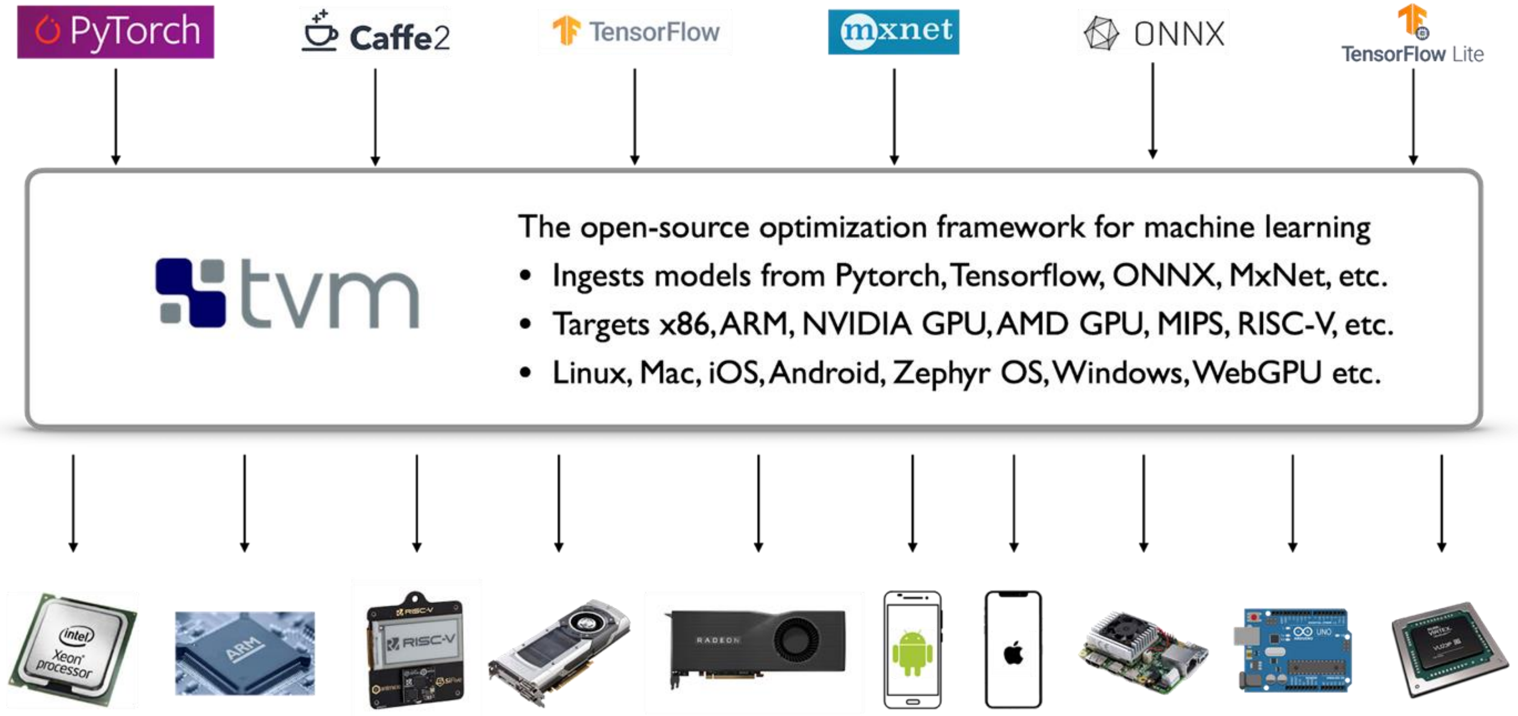
Let the Deployment Challenge Begin!



PyTorch	?	?	?	?	?	?	?	?	?	?
Caffe2	?	⊖ ☐	?	?	?	?	?	?	?	?
TensorFlow	?	?	?	?	?	?	?	?	?	?
mxnet	?	?	?	?	?	?	?	?	?	?
ONNX	?	?	?	?	?	?	?	?	?	?
TensorFlow Lite	?	?	?	?	?	?	?	?	?	?

OctoML

Introducing TVM



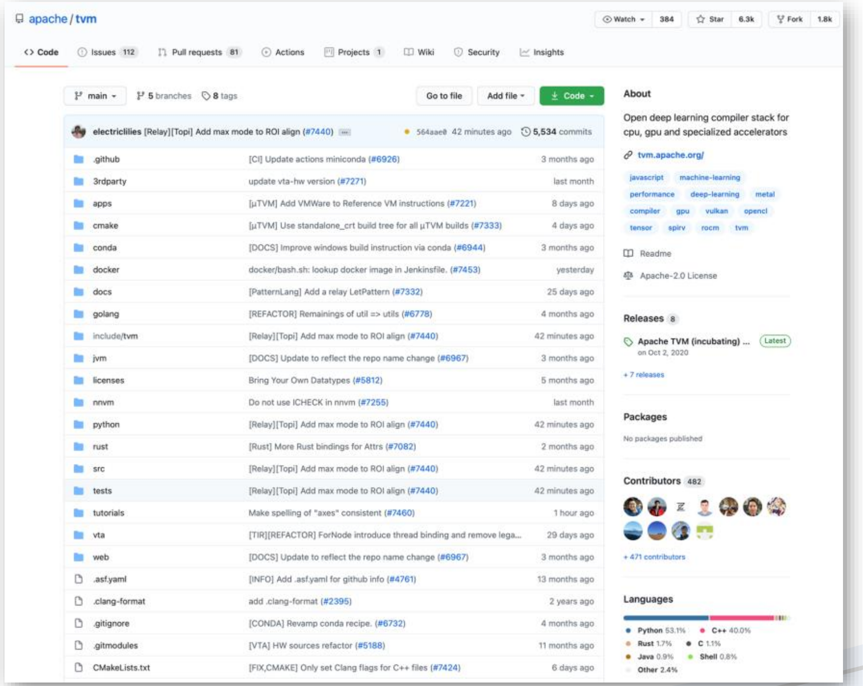
The open-source optimization framework for machine learning

- Ingests models from Pytorch, Tensorflow, ONNX, MxNet, etc.
- Targets x86, ARM, NVIDIA GPU, AMD GPU, MIPS, RISC-V, etc.
- Linux, Mac, iOS, Android, Zephyr OS, Windows, WebGPU etc.

OctoML

Introducing TVM

- TVM is an open source optimizing compiler framework for machine learning.
- An official Apache project.
- Community owned.
- Active discussion forums.
- Regular meetups and conferences.


apache / tvm

electriclilies [Relay][Topi] Add max mode to ROI align (#7440) 564aae0 42 minutes ago 5,534 commits

.github	[CI] Update actions miniconda (#6926)	3 months ago
3rdparty	update vta-hw version (#7271)	last month
apps	[µTVM] Add VMWare to Reference VM instructions (#7221)	8 days ago
cmake	[µTVM] Use standalone_crt build tree for all µTVM builds (#7333)	4 days ago
conda	[DOCS] Improve windows build instruction via conda (#6944)	3 months ago
docker	docker/bash.sh: lookup docker image in Jenkinsfile. (#7453)	yesterday
docs	[PatternLang] Add a relay LetPattern (#7332)	25 days ago
golang	[REFACTOR] Remainings of util => utils (#6778)	4 months ago
include/tvm	[Relay][Topi] Add max mode to ROI align (#7440)	42 minutes ago
jvm	[DOCS] Update to reflect the repo name change (#6967)	3 months ago
licenses	Bring Your Own Datatypes (#5812)	5 months ago
nnvm	Do not use ICHECK in nnvm (#7255)	last month
python	[Relay][Topi] Add max mode to ROI align (#7440)	42 minutes ago
rust	[Rust] More Rust bindings for Attrs (#7082)	2 months ago
src	[Relay][Topi] Add max mode to ROI align (#7440)	42 minutes ago
tests	[Relay][Topi] Add max mode to ROI align (#7440)	42 minutes ago
tutorials	Make spelling of "axes" consistent (#7460)	1 hour ago
vta	[TIR][REFACTOR] ForNode introduce thread binding and remove lega...	29 days ago
web	[DOCS] Update to reflect the repo name change (#6967)	3 months ago
.asf.yaml	[INFO] Add .asf.yaml for github info (#4761)	13 months ago
.clang-format	add .clang-format (#2395)	2 years ago
.gitignore	[CONDA] Revamp conda recipe. (#6732)	4 months ago
.gitmodules	[VTA] HW sources refactor (#5188)	11 months ago
CMakeLists.txt	[FIX,CMAKE] Only set Clang flags for C++ files (#7424)	6 days ago

About

Open deep learning compiler stack for cpu, gpu and specialized accelerators

tvm.apache.org/

javascript machine-learning performance deep-learning metal compiler gpu vulkan opencl tensor spirv rocm tvm

Readme

Apache-2.0 License

Releases 8

Apache TVM (incubating) ... Latest
on Oct 2, 2020

+ 7 releases

Packages

No packages published

Contributors 482

+ 471 contributors

Languages

- Python 53.1%
- C++ 40.0%
- Rust 1.7%
- C 1.1%
- Java 0.9%
- Shell 0.8%
- Other 2.4%

OctoML

Problems TVM Addresses

- Portability: When there are limited hardware options to deploy your model to.
- Efficiency: When you need to make your model run effectively on a target platform.
- Software Support: When you need to build a new software stack for your hardware system.

OctoML

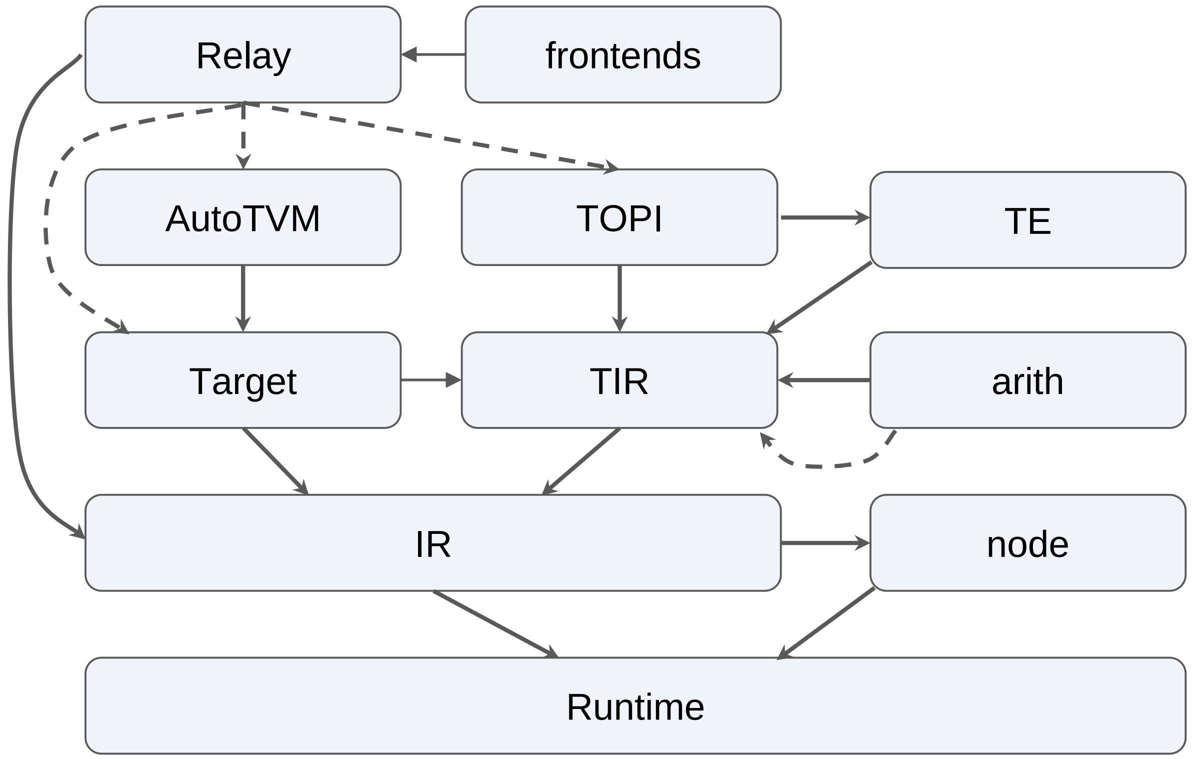
Why should you use TVM?

How does TVM work?

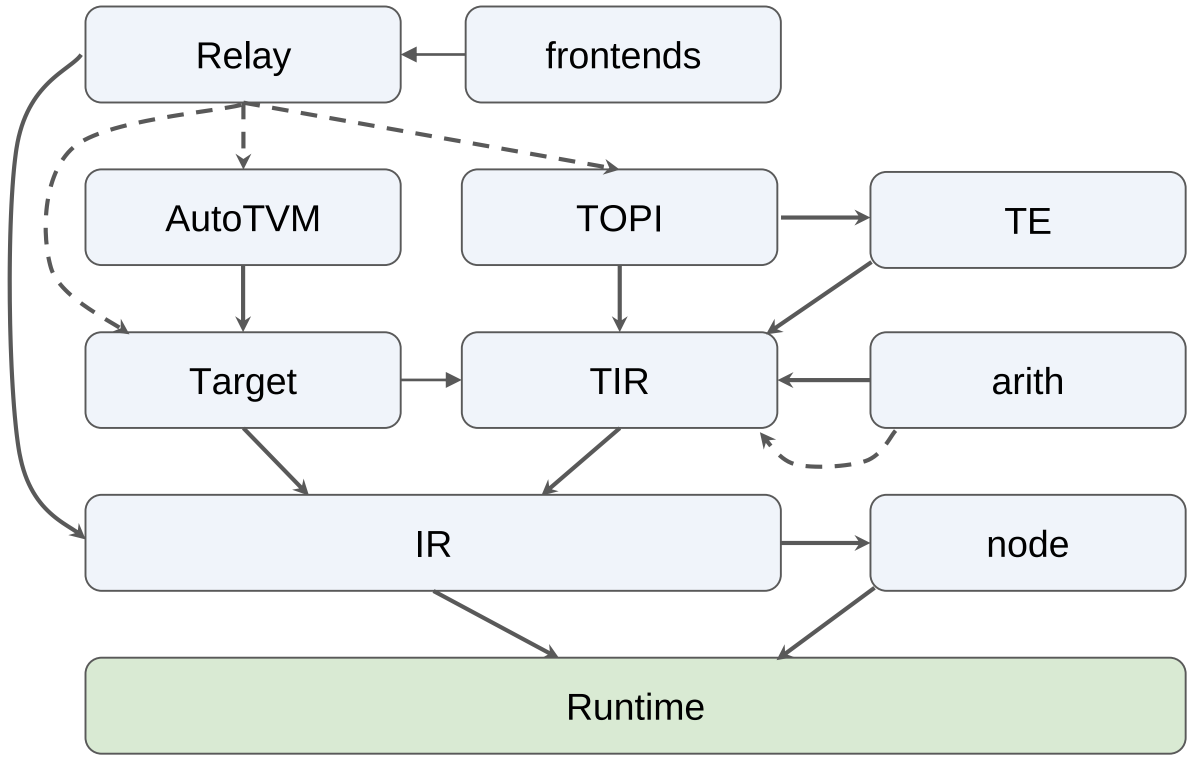
How do you install TVM?

How do you use TVMC?

A Quick Overview of TVM

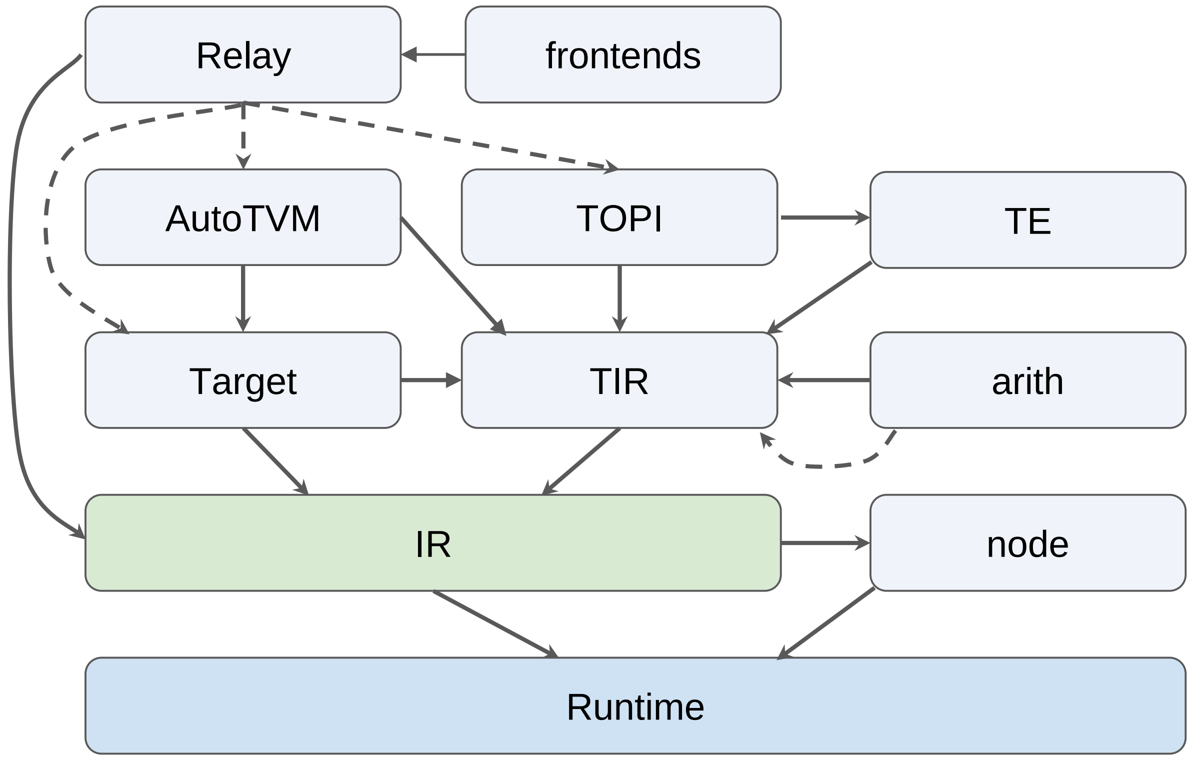


A Quick Overview of TVM - Runtime



The Runtime is the foundation of the TVM stack. It provides the foundation to load and run compiled TVM artifacts.

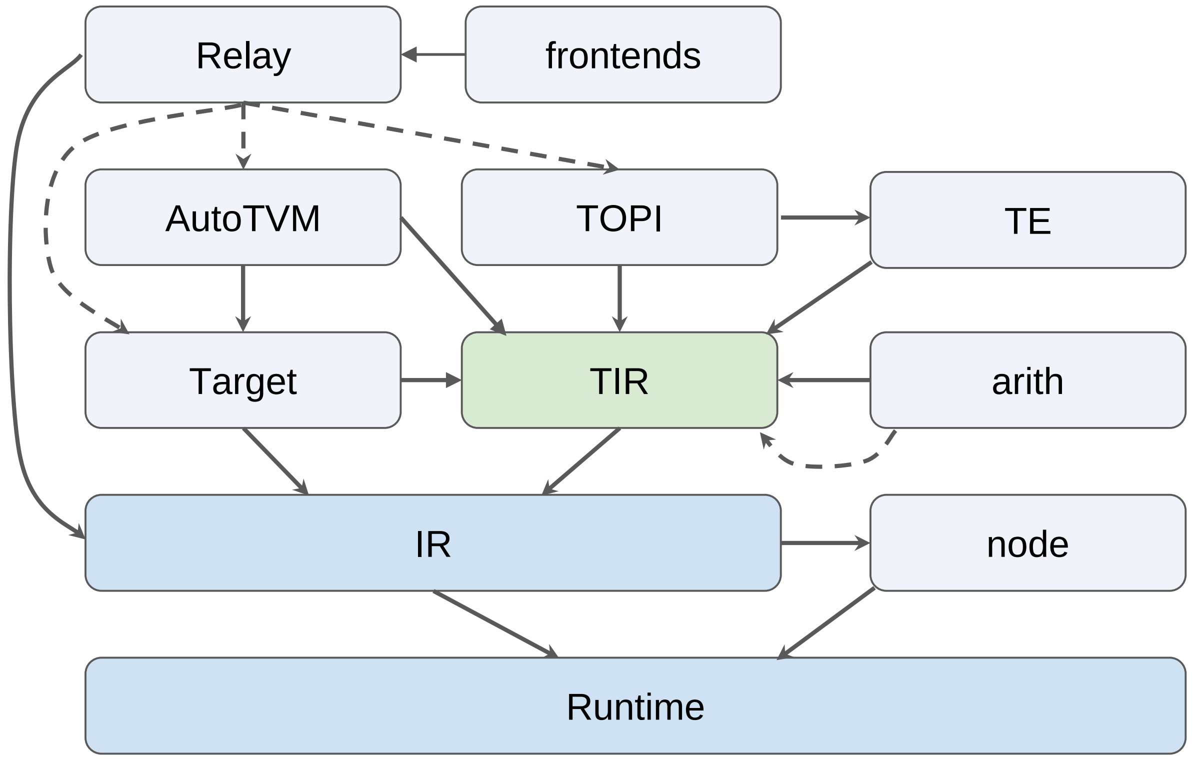
A Quick Overview of TVM - IR



The IR layer of TVM is the low level compiled Intermediate Representation of a computation, with unified data structures and interfaces for all function variants.

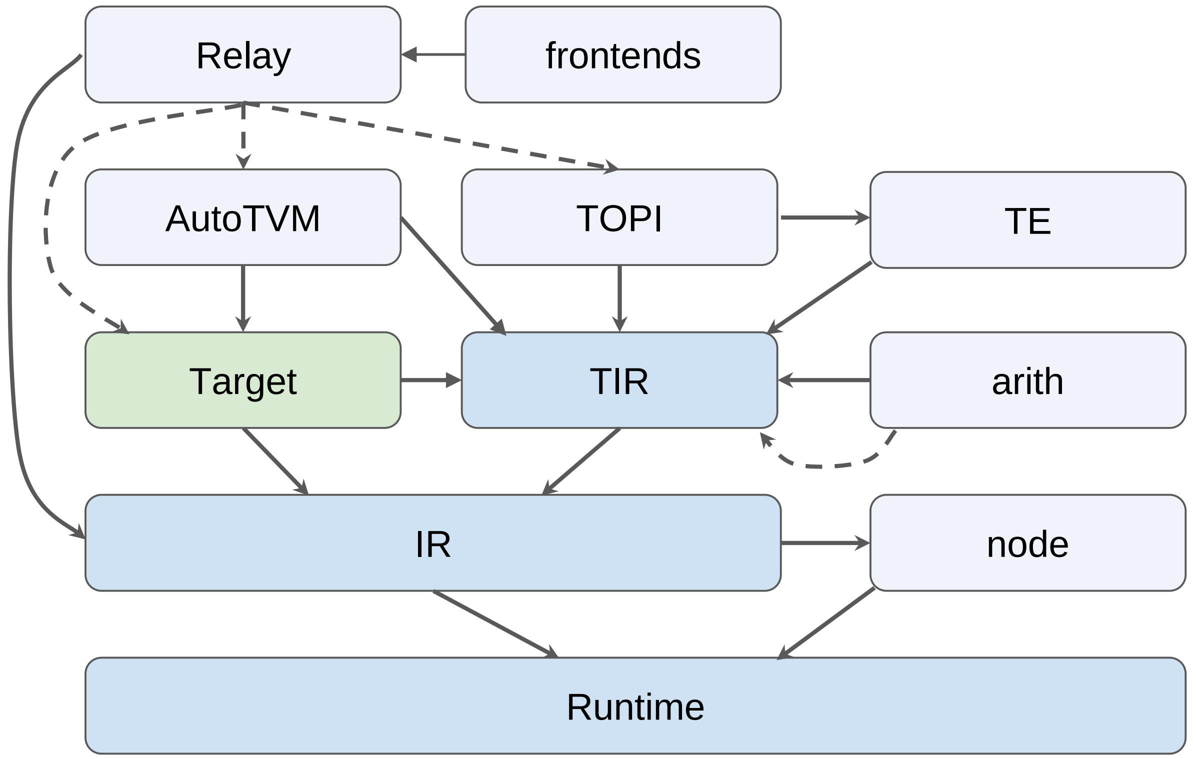
OctoML

A Quick Overview of TVM - TIR



The TIR module has low level language invariants that represent functions which can be transformed to equivalent functions by multiple passes.

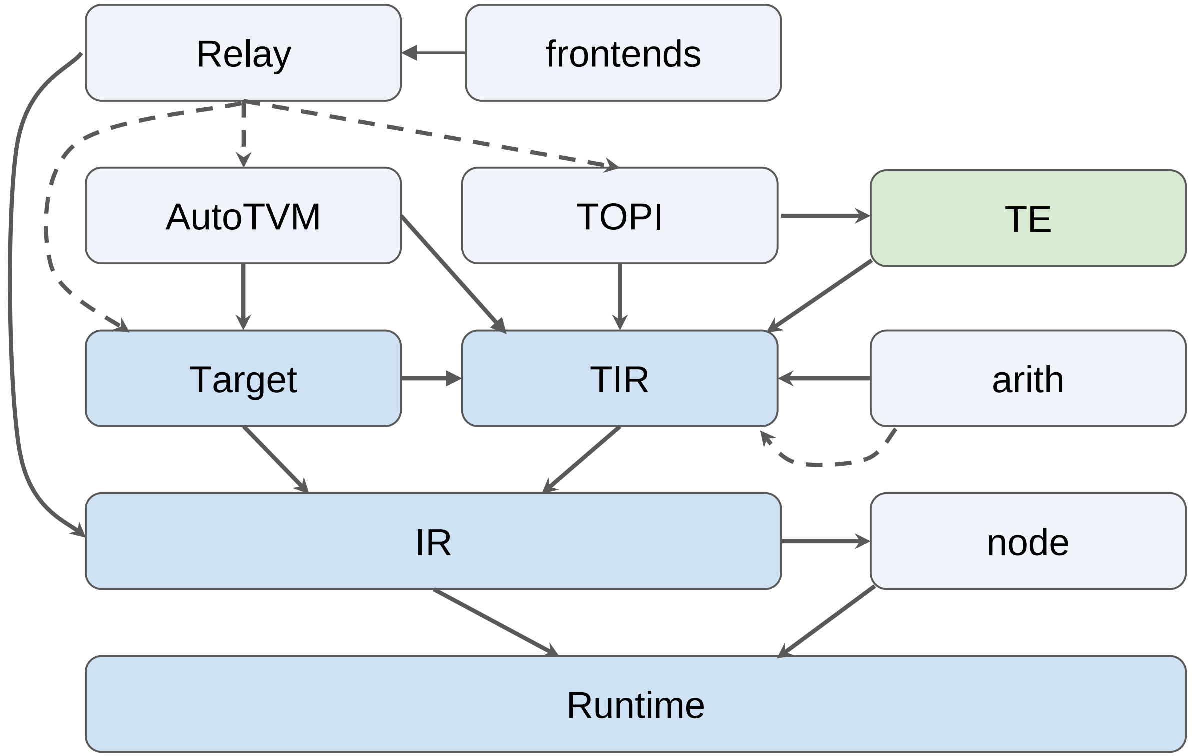
A Quick Overview of TVM - Target



The Target module contains the code that allows an IR function to be transformed into a runtime object. It allows for TVM to target different architectures.

OctoML

A Quick Overview of TVM - TE

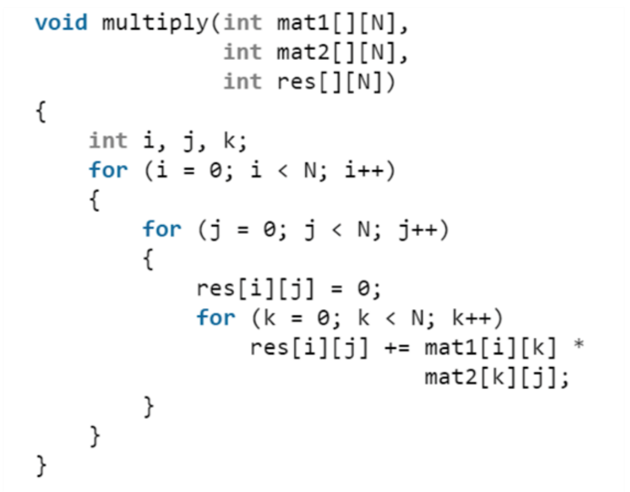


The Tensor Expression (TE) module is a domain specific functional language that allows for the templated expression of tensor computation, which can be transformed in TIR passes using schedule operators.

OctoML

An Example Operator: Matrix Multiplication

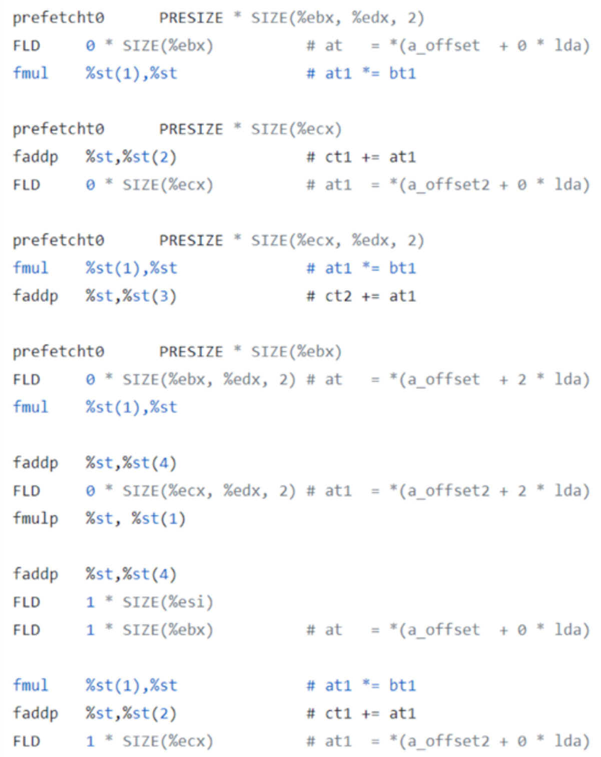
Naive Approach



```c
void multiply(int mat1[][N],
              int mat2[][N],
              int res[][N])
{
    int i, j, k;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            res[i][j] = 0;
            for (k = 0; k < N; k++)
                res[i][j] += mat1[i][k] *
                             mat2[k][j];
        }
    }
}
```

10-100x faster

Optimized Approach (snippet from 1000s lines of code)



```
prefetcht0      PRESIZE * SIZE(%ebx, %edx, 2)
FLD     0 * SIZE(%ebx)          # at    = *(a_offset  + 0 * lda)
fmul    %st(1),%st              # at1 *= bt1

prefetcht0      PRESIZE * SIZE(%ecx)
faddp   %st,%st(2)              # ct1 += at1
FLD     0 * SIZE(%ecx)          # at1  = *(a_offset2 + 0 * lda)

prefetcht0      PRESIZE * SIZE(%ecx, %edx, 2)
fmul    %st(1),%st              # at1 *= bt1
faddp   %st,%st(3)              # ct2 += at1

prefetcht0      PRESIZE * SIZE(%ebx)
FLD     0 * SIZE(%ebx, %edx, 2) # at    = *(a_offset  + 2 * lda)
fmul    %st(1),%st

faddp   %st,%st(4)
FLD     0 * SIZE(%ecx, %edx, 2) # at1  = *(a_offset2 + 2 * lda)
fmulp   %st, %st(1)

faddp   %st,%st(4)
FLD     1 * SIZE(%esi)
FLD     1 * SIZE(%ebx)          # at    = *(a_offset  + 0 * lda)

fmul    %st(1),%st              # at1 *= bt1
faddp   %st,%st(2)              # ct1 += at1
FLD     1 * SIZE(%ecx)          # at1  = *(a_offset2 + 0 * lda)
```

Basic Matrix Multiplication in TE and TIR

Tensor-Expression DSL defines the algorithm and the schedule

```
# Algorithm
k = te.reduce_axis((0, K), "k")
A = te.placeholder((M, K), name="A")
B = te.placeholder((K, N), name="B")
C = te.compute((M, N), lambda x, y: te.sum(A[x, k] * B[k, y], axis=k), name="C")

# Default schedule
s = te.create_schedule(C.op)
```

vanilla schedule

Tensor-Level IR (TIR) program defines the low-level implementation (can be compiled down to LLVM IR, CUDA, OpenCL et.)

```
primfn(A_1: handle, B_1: handle, C_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {B: Buffer(B_2: Pointer(float32), float32, [1024, 1024], []),
             C: Buffer(C_2: Pointer(float32), float32, [1024, 1024], []),
             A: Buffer(A_2: Pointer(float32), float32, [1024, 1024], [])}
  buffer_map = {A_1: A, B_1: B, C_1: C} {
  for (x: int32, 0, 1024) {
    for (y: int32, 0, 1024) {
      C_2[((x*1024) + y)] = 0f32
      for (k: int32, 0, 1024) {
        C_2[((x*1024) + y)] = ((float32*)C_2[((x*1024) + y)] + ((float32*)A_2[((x*1024) + k)]*(float32*)B_2[((k*1024) + y)]))
      }
    }
  }
}
```

3 nested loops

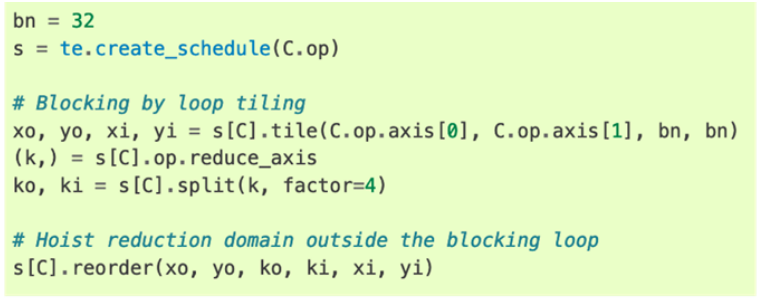
OctoML

Tiling and Reordering of MatMul in TE and TIR

vanilla schedule

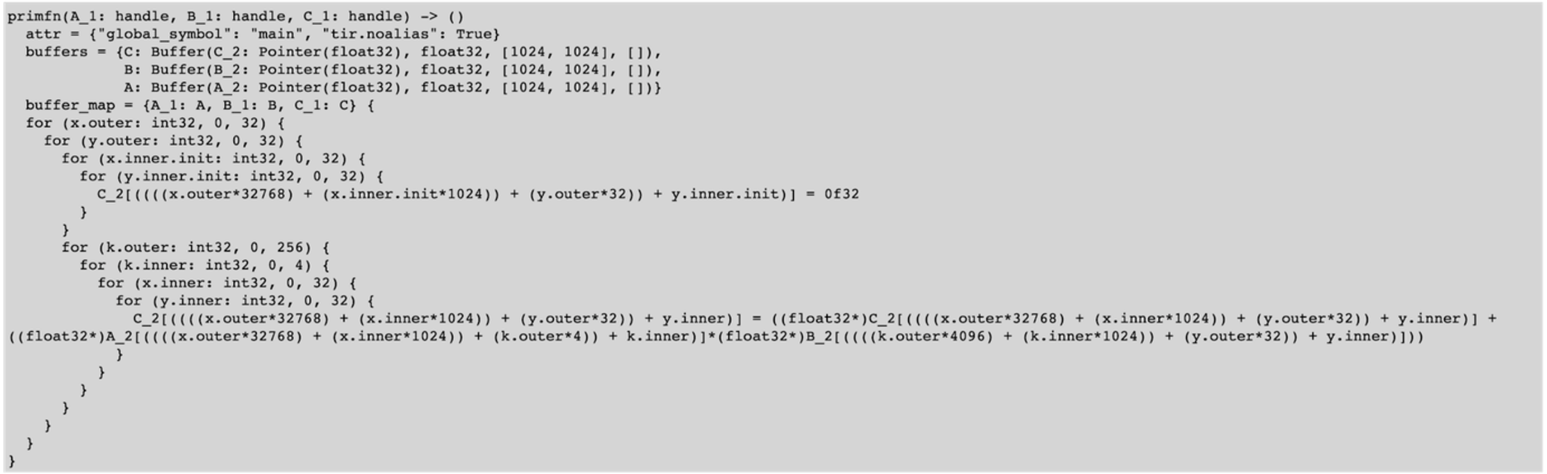
compute tiling

split reduction axis



```
bn = 32
s = te.create_schedule(C.op)

# Blocking by loop tiling
xo, yo, xi, yi = s[C].tile(C.op.axis[0], C.op.axis[1], bn, bn)
(k,) = s[C].op.reduce_axis
ko, ki = s[C].split(k, factor=4)

# Hoist reduction domain outside the blocking loop
s[C].reorder(xo, yo, ko, ki, xi, yi)
```

6 nested loops



```
primfn(A_1: handle, B_1: handle, C_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {C: Buffer(C_2: Pointer(float32), float32, [1024, 1024], []),
             B: Buffer(B_2: Pointer(float32), float32, [1024, 1024], []),
             A: Buffer(A_2: Pointer(float32), float32, [1024, 1024], [])}
  buffer_map = {A_1: A, B_1: B, C_1: C} {
  for (x.outer: int32, 0, 32) {
    for (y.outer: int32, 0, 32) {
      for (x.inner.init: int32, 0, 32) {
        for (y.inner.init: int32, 0, 32) {
          C_2[((((x.outer*32768) + (x.inner.init*1024)) + (y.outer*32)) + y.inner.init)] = 0f32
        }
      }
      for (k.outer: int32, 0, 256) {
        for (k.inner: int32, 0, 4) {
          for (x.inner: int32, 0, 32) {
            for (y.inner: int32, 0, 32) {
              C_2[((((x.outer*32768) + (x.inner*1024)) + (y.outer*32)) + y.inner)] = ((float32*)C_2[((((x.outer*32768) + (x.inner*1024)) + (y.outer*32)) + y.inner)] +
((float32*)A_2[((((x.outer*32768) + (x.inner*1024)) + (k.outer*4)) + k.inner)]*(float32*)B_2[((((k.outer*4096) + (k.inner*1024)) + (y.outer*32)) + y.inner)]))
            }
          }
        }
      }
    }
  }
}
```

OctoML

Vectorization of Matrix Multiplication in TE and TIR

vanilla schedule

compute tiling

split reduction axis

vectorize

```
s = te.create_schedule(C.op)
xo, yo, xi, yi = s[C].tile(C.op.axis[0], C.op.axis[1], bn, bn)
(k,) = s[C].op.reduce_axis
ko, ki = s[C].split(k, factor=4)

s[C].reorder(xo, yo, ko, ki, xi, yi)

# Vectorization
s[C].vectorize(yi)
```

5 nested loops

vectorization

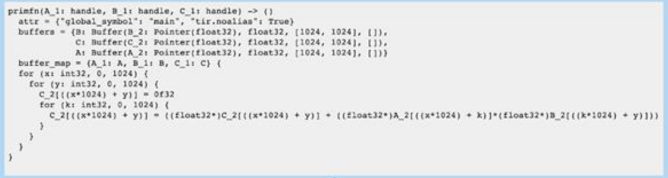
```
primfn(A_1: handle, B_1: handle, C_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {C: Buffer(C_2: Pointer(float32), float32, [1024, 1024], []),
             B: Buffer(B_2: Pointer(float32), float32, [1024, 1024], []),
             A: Buffer(A_2: Pointer(float32), float32, [1024, 1024], [])}
  buffer_map = {A_1: A, B_1: B, C_1: C} {
  for (x.outer: int32, 0, 32) {
    for (y.outer: int32, 0, 32) {
      for (x.inner.init: int32, 0, 32) {
        C_2[ramp((((x.outer*32768) + (x.inner.init*1024)) + (y.outer*32)), 1, 32)] = broadcast(0f32, 32)
      }
      for (k.outer: int32, 0, 256) {
        for (k.inner: int32, 0, 4) {
          for (x.inner: int32, 0, 32) {
            C_2[ramp((((x.outer*32768) + (x.inner*1024)) + (y.outer*32)), 1, 32)] = ((float32x32*)C_2[ramp((((x.outer*32768) + (x.inner*1024)) + (y.outer*32)), 1, 32)] +
(broadcast((float32*)A_2[((((x.outer*32768) + (x.inner*1024)) + (k.outer*4)) + k.inner)], 32)*(float32x32*)B_2[ramp((((k.outer*4096) + (k.inner*1024)) + (y.outer*32)), 1, 32)]))
          }
        }
      }
    }
  }
}
```

OctoML

Optimized Matrix Multiplication in TE and TIR

Before

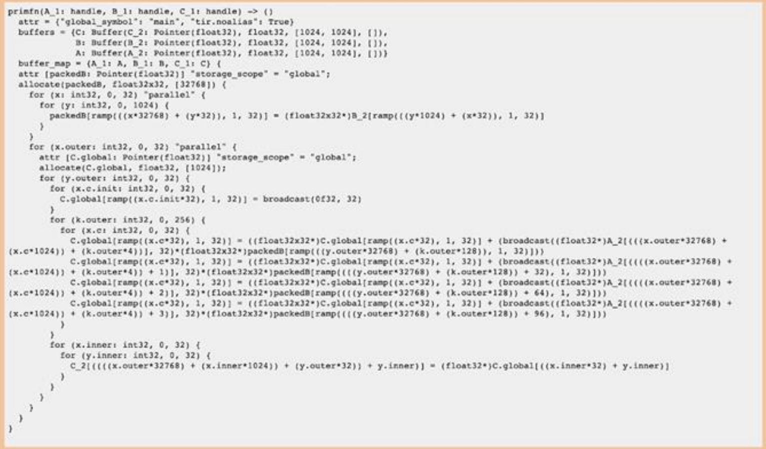
```
s = te.create_schedule(C.op)
```



```
primfn(A_1: handle, B_1: handle, C_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {B: Buffer(B_2: Pointer(float32), float32, [1024, 1024], []),
             C: Buffer(C_2: Pointer(float32), float32, [1024, 1024], []),
             A: Buffer(A_2: Pointer(float32), float32, [1024, 1024], [])}
  buffer_map = {A_1: A, B_1: B, C_1: C} {
  for (x: int32, 0, 1024) {
    for (y: int32, 0, 1024) {
      C_2[((x*1024) + y)] = 0f32
      for (k: int32, 0, 1024) {
        C_2[((x*1024) + y)] = ((float32*)C_2[((x*1024) + y)] + ((float32*)A_2[((x*1024) + k)]*(float32*)B_2[((k*1024) + y)]))
      }
    }
  }
}
```

After: 200x faster

```
s = te.create_schedule(C.op)

CC = s.cache_write(C, "global")

xo, yo, xi, yi = s[C].tile(C.op.axis[0], C.op.axis[1], bn, bn)

s[CC].compute_at(s[C], yo)

xc, yc = s[CC].op.axis

(k,) = s[CC].op.reduce_axis
ko, ki = s[CC].split(k, factor=4)
s[CC].reorder(ko, xc, ki, yc)
s[CC].unroll(ki)
s[CC].vectorize(yc)

# parallel
s[C].parallel(xo)

x, y, z = s[packedB].op.axis
s[packedB].vectorize(z)
s[packedB].parallel(x)
```

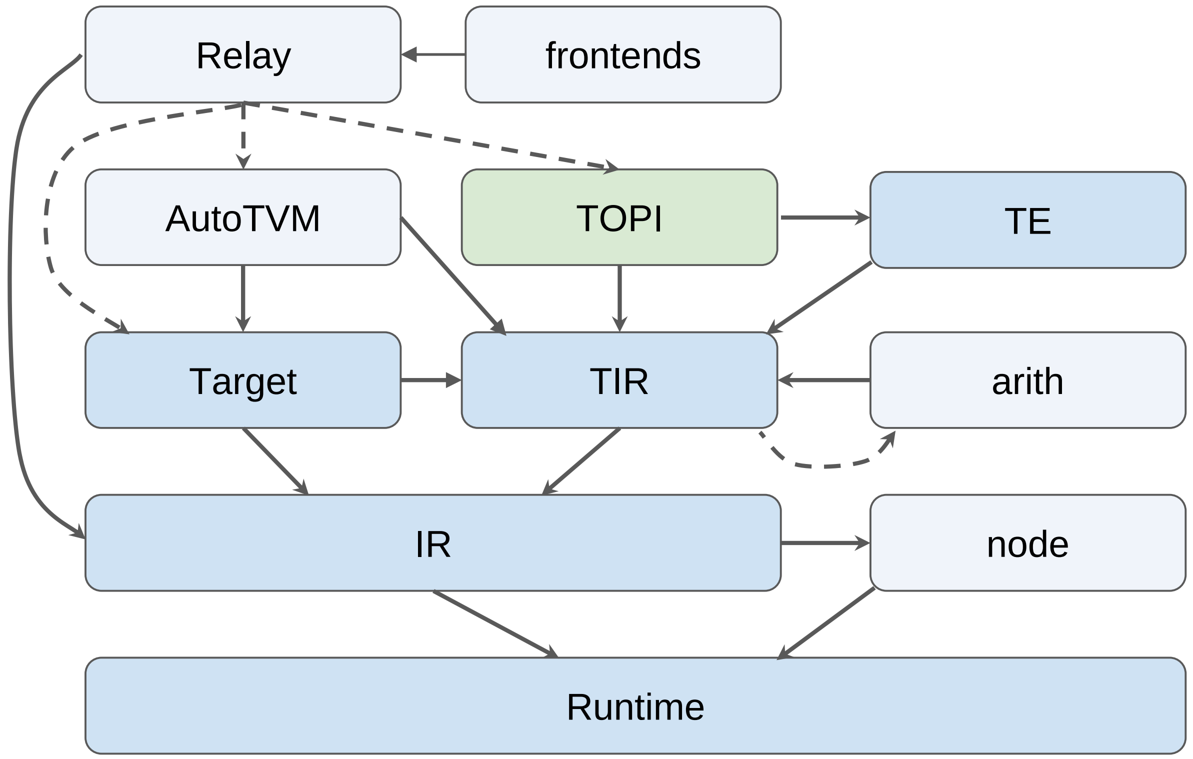


```
primfn(A_1: handle, B_1: handle, C_1: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {C: Buffer(C_2: Pointer(float32), float32, [1024, 1024], []),
             B: Buffer(B_2: Pointer(float32), float32, [1024, 1024], []),
             A: Buffer(A_2: Pointer(float32), float32, [1024, 1024], [])}
  buffer_map = {A_1: A, B_1: B, C_1: C} {
  attr [packedB: Pointer(float32)] "storage_scope" = "global";
  allocate(packedB, float32x32, [32768]) {
    for (x: int32, 0, 32) "parallel" {
      for (y: int32, 0, 1024) {
        packedB[ramp(((x*32768) + (y*32)), 1, 32)] = (float32x32*)B_2[ramp(((y*1024) + (x*32)), 1, 32)]
      }
    }
    for (x.outer: int32, 0, 32) "parallel" {
      attr [C.global: Pointer(float32)] "storage_scope" = "global";
      allocate(C.global, float32, [1024]);
      for (y.outer: int32, 0, 32) {
        for (x.c.init: int32, 0, 32) {
          C.global[ramp((x.c.init*32), 1, 32)] = broadcast(0f32, 32)
        }
        for (k.outer: int32, 0, 256) {
          for (x.c: int32, 0, 32) {
            C.global[ramp((x.c*32), 1, 32)] = ((float32x32*)C.global[ramp((x.c*32), 1, 32)] + (broadcast((float32*)A_2[(((x.outer*32768) +
(x.c*1024)) + (k.outer*4))], 32)*(float32x32*)packedB[ramp(((y.outer*32768) + (k.outer*128)), 1, 32)]))
            C.global[ramp((x.c*32), 1, 32)] = ((float32x32*)C.global[ramp((x.c*32), 1, 32)] + (broadcast((float32*)A_2[((((x.outer*32768) +
(x.c*1024)) + (k.outer*4)) + 1)], 32)*(float32x32*)packedB[ramp((((y.outer*32768) + (k.outer*128)) + 32), 1, 32)]))
            C.global[ramp((x.c*32), 1, 32)] = ((float32x32*)C.global[ramp((x.c*32), 1, 32)] + (broadcast((float32*)A_2[((((x.outer*32768) +
(x.c*1024)) + (k.outer*4)) + 2)], 32)*(float32x32*)packedB[ramp((((y.outer*32768) + (k.outer*128)) + 64), 1, 32)]))
            C.global[ramp((x.c*32), 1, 32)] = ((float32x32*)C.global[ramp((x.c*32), 1, 32)] + (broadcast((float32*)A_2[((((x.outer*32768) +
(x.c*1024)) + (k.outer*4)) + 3)], 32)*(float32x32*)packedB[ramp((((y.outer*32768) + (k.outer*128)) + 96), 1, 32)]))
          }
        }
        for (x.inner: int32, 0, 32) {
          for (y.inner: int32, 0, 32) {
            C_2[((((x.outer*32768) + (x.inner*1024)) + (y.outer*32)) + y.inner)] = (float32*)C.global[((x.inner*32) + y.inner)]
          }
        }
      }
    }
  }
}
```

Optimized Matrix Multiplication Summary

- With 13 lines of scheduling code, we can increase performance by about 200x
  - Tiling, loop permutations lead to better cache hit rates
  - Array packing to turn non-continuous access patterns into continuous pattern
  - Vectorizations takes advantage of special hardware instructions
  - Thread-level parallelization makes use of all of the cores

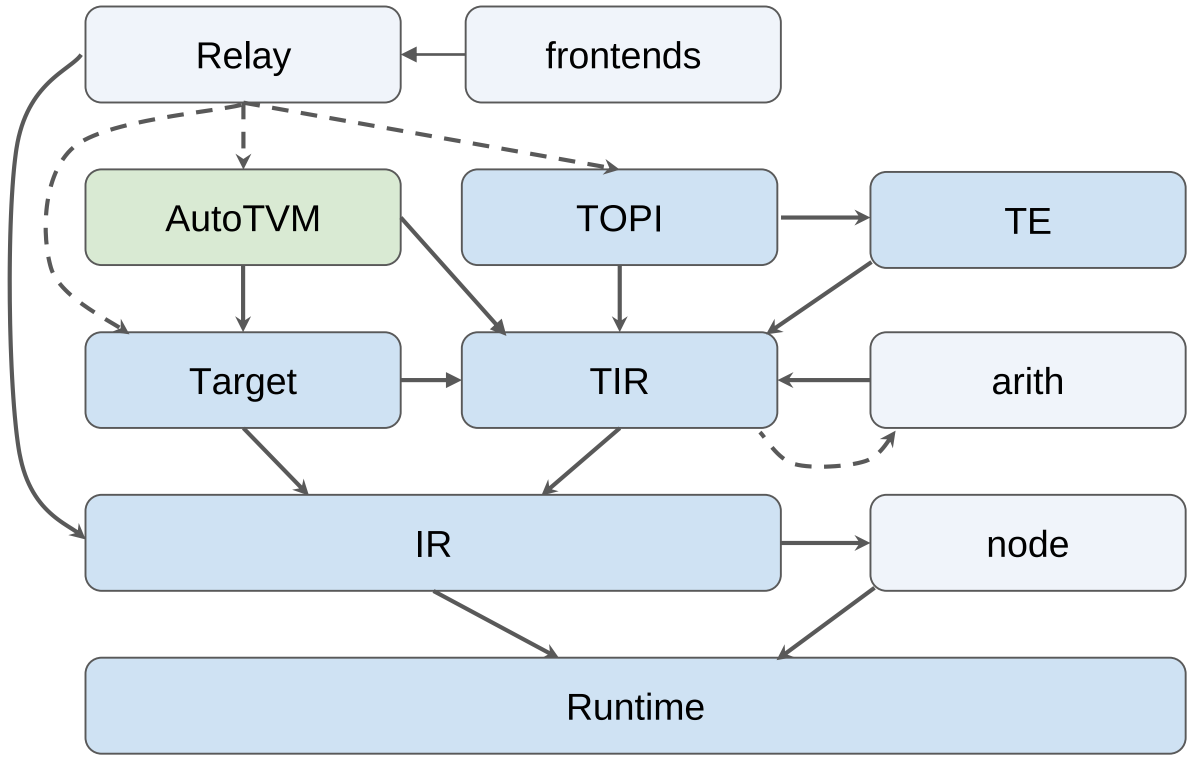
A Quick Overview of TVM - TOPI



The Tensor Operator Inventory (TOPI) is a collection of predefined TE templates covering a range of common operators for a range of platforms.

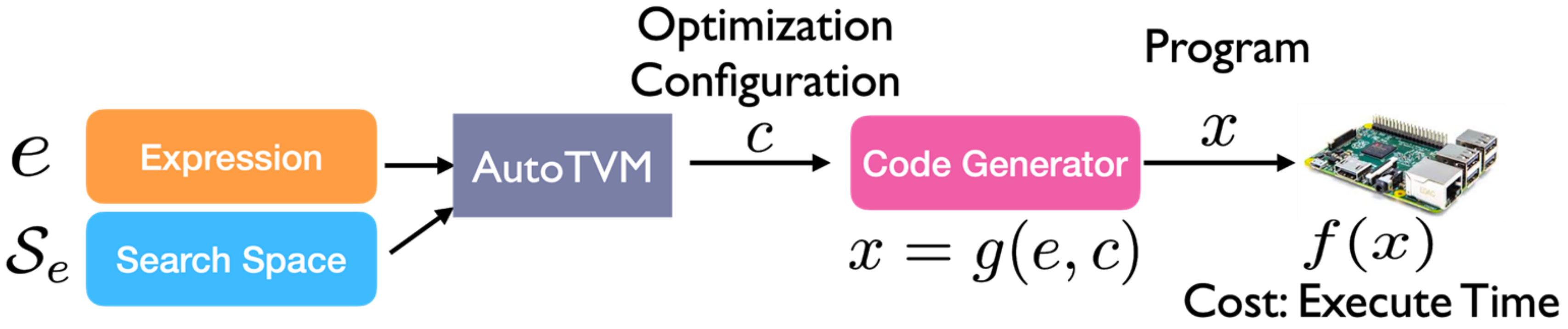
OctoML

A Quick Overview of TVM - AutoTVM



AutoTVM and AutoScheduler are both components that automate the search for an optimized schedule. Includes tuning records, cost models, feature extraction, and search policies.

AutoTVM for Finding Optimal Schedules

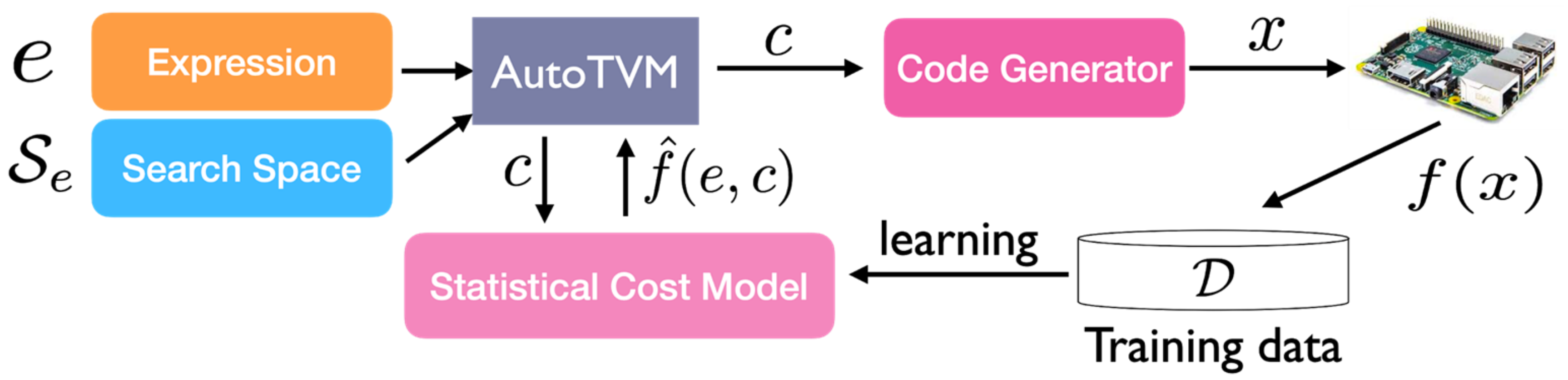


Objective $argmin_{c \in S_e} f(g(e, c))$

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

AutoTVM for Finding Optimal Schedules with ML

Use machine learning to learn a statistical cost model

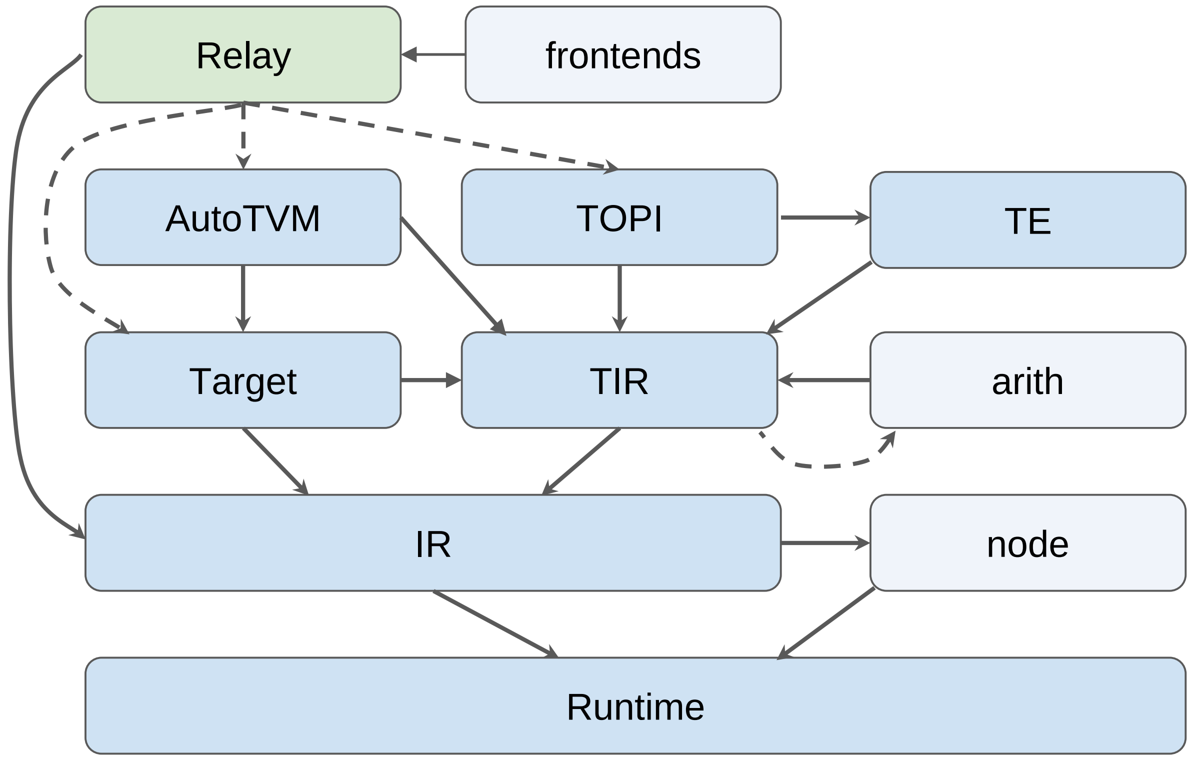


Benefit: Automatically adapt to hardware type

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

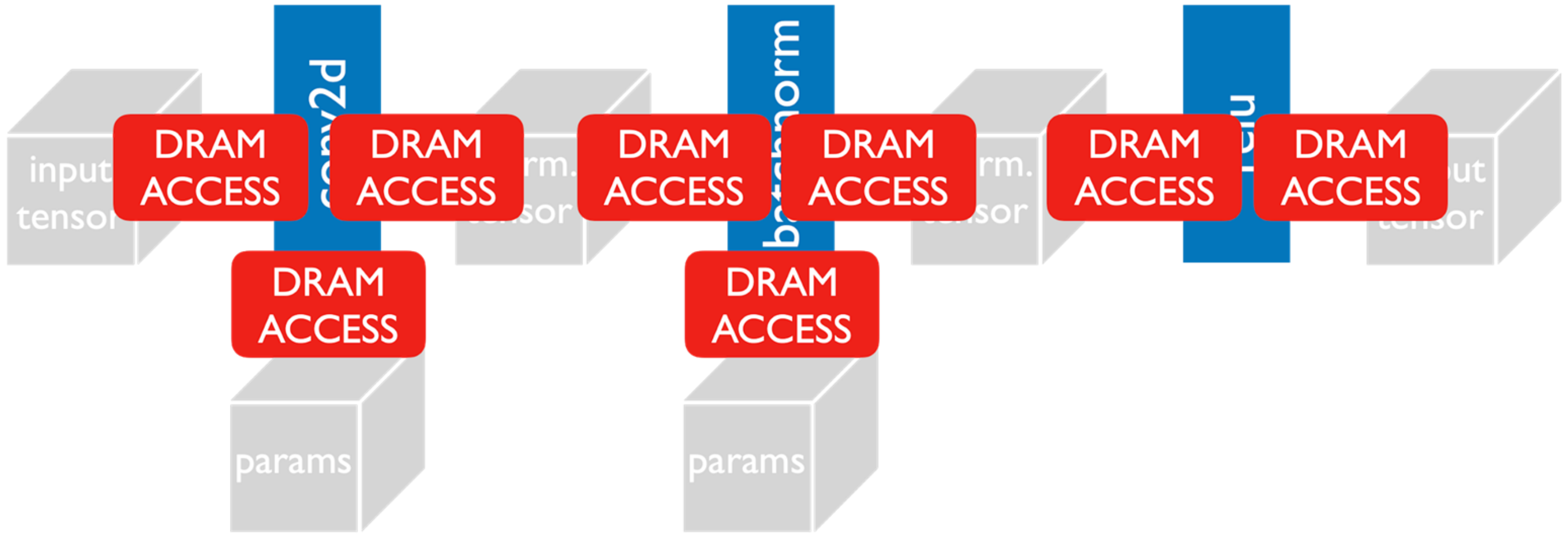
OctoML

A Quick Overview of TVM - Relay



Relay is a high level functional intermediate representation that is used to represent full models. It supports different ‘dialects’ for different types of optimizations (quantization, memory, dynamic vms).

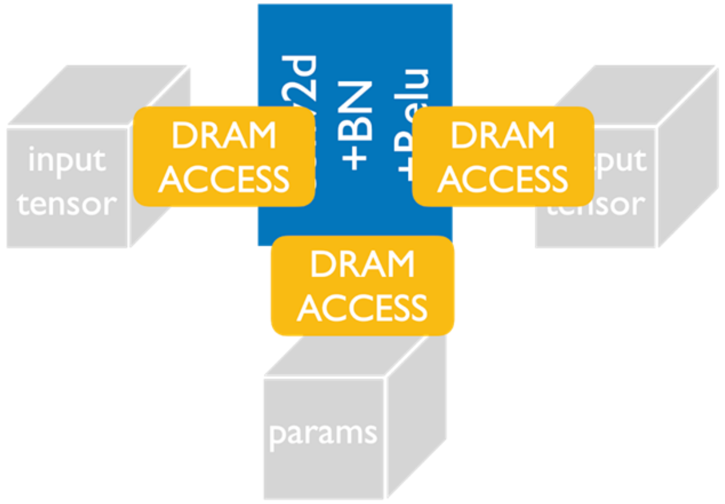
Graph Level Optimization with Relay



Are these many DRAM accesses really necessary?

OctoML

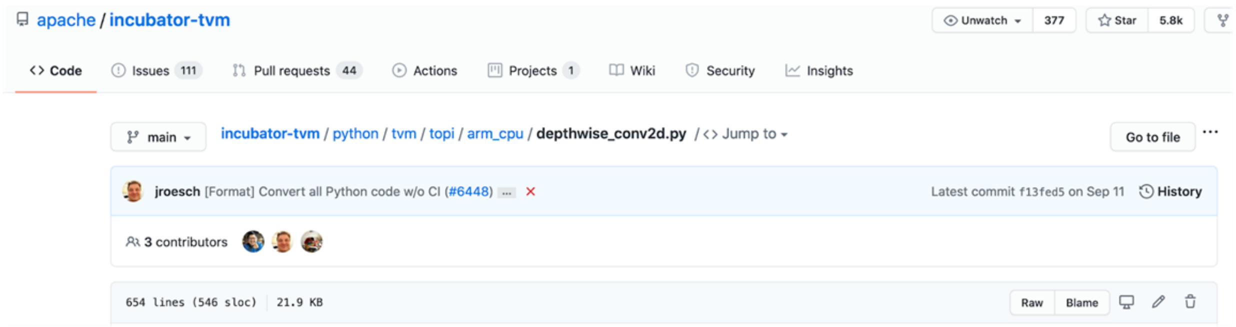
Graph Level Optimization with Relay - Fusion



- The idea is to fuse multiple operators into a single operator to minimize memory access
- If you have to rely on an operator library, you need to add implementations for fused operators! That's a long list of operators.
- Thankfully with more flexible code-generation approaches like TVM, we can generate fused kernels on demand.

OctoML

Results with TVM


apache / incubator-tvm

Unwatch 377 Star 5.8k

Code Issues 111 Pull requests 44 Actions Projects 1 Wiki Security Insights

main incubator-tvm / python / tvm / topi / arm_cpu / depthwise_conv2d.py / <> Jump to

Go to file

jroesch [Format] Convert all Python code w/o CI (#6448) ... Latest commit f13fed5 on Sep 11 History

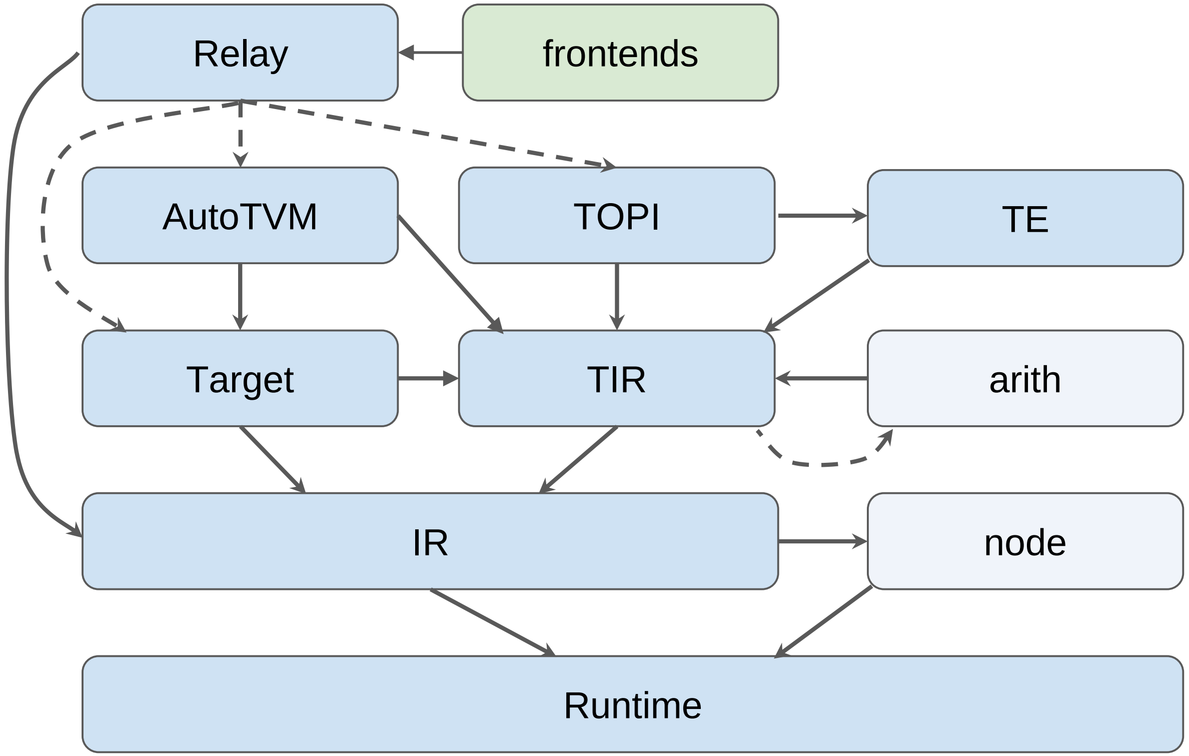
3 contributors

```
654 lines (546 sloc) | 21.9 KB
```

Raw Blame

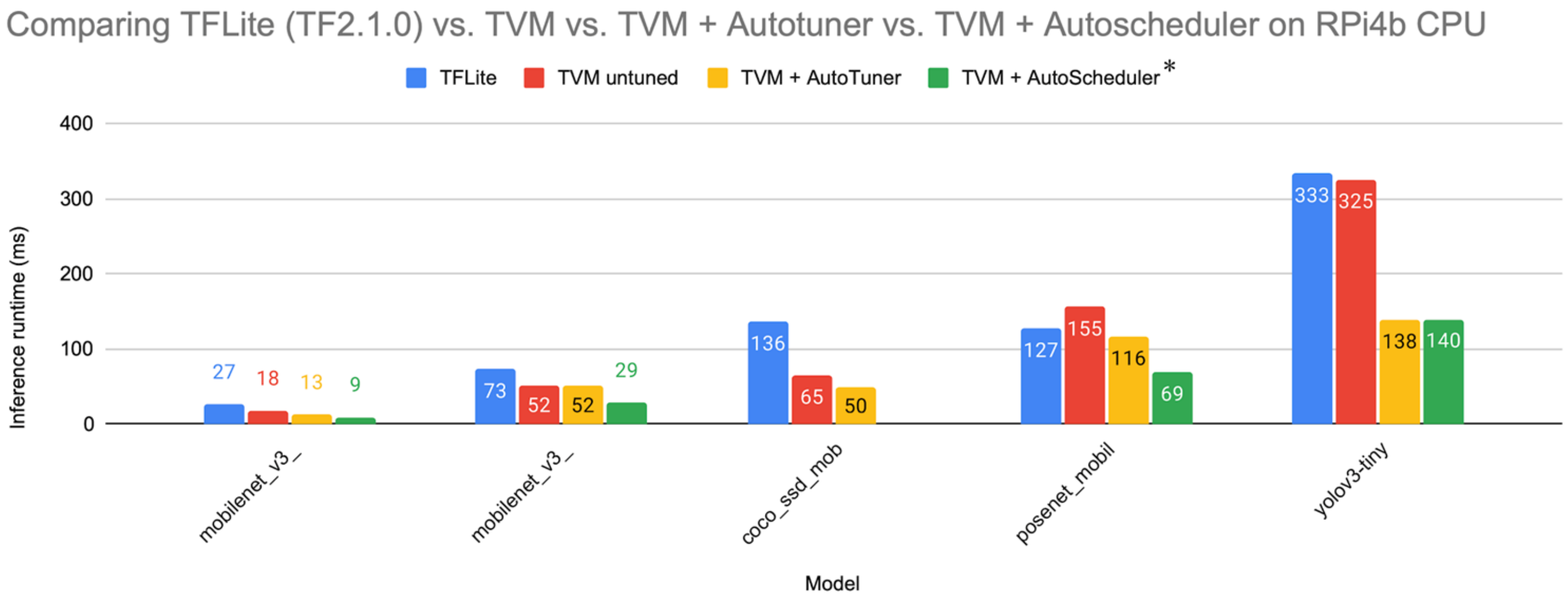
- TVM definition and schedule implementation for depthwise conv2d is less than 700 lines long (much less than 13k lines)
  - This covers all kernel sizes (not just 3x3), data type implementation (not just uint8)

A Quick Overview of TVM - frontends



TVM includes a number of different frontends to ingest models from different frameworks.

Results on RPi with TVM



* Ansor: Generating High-Performance Tensor Programs for Deep Learning, Zheng et al. OSDI 2020

OctoML

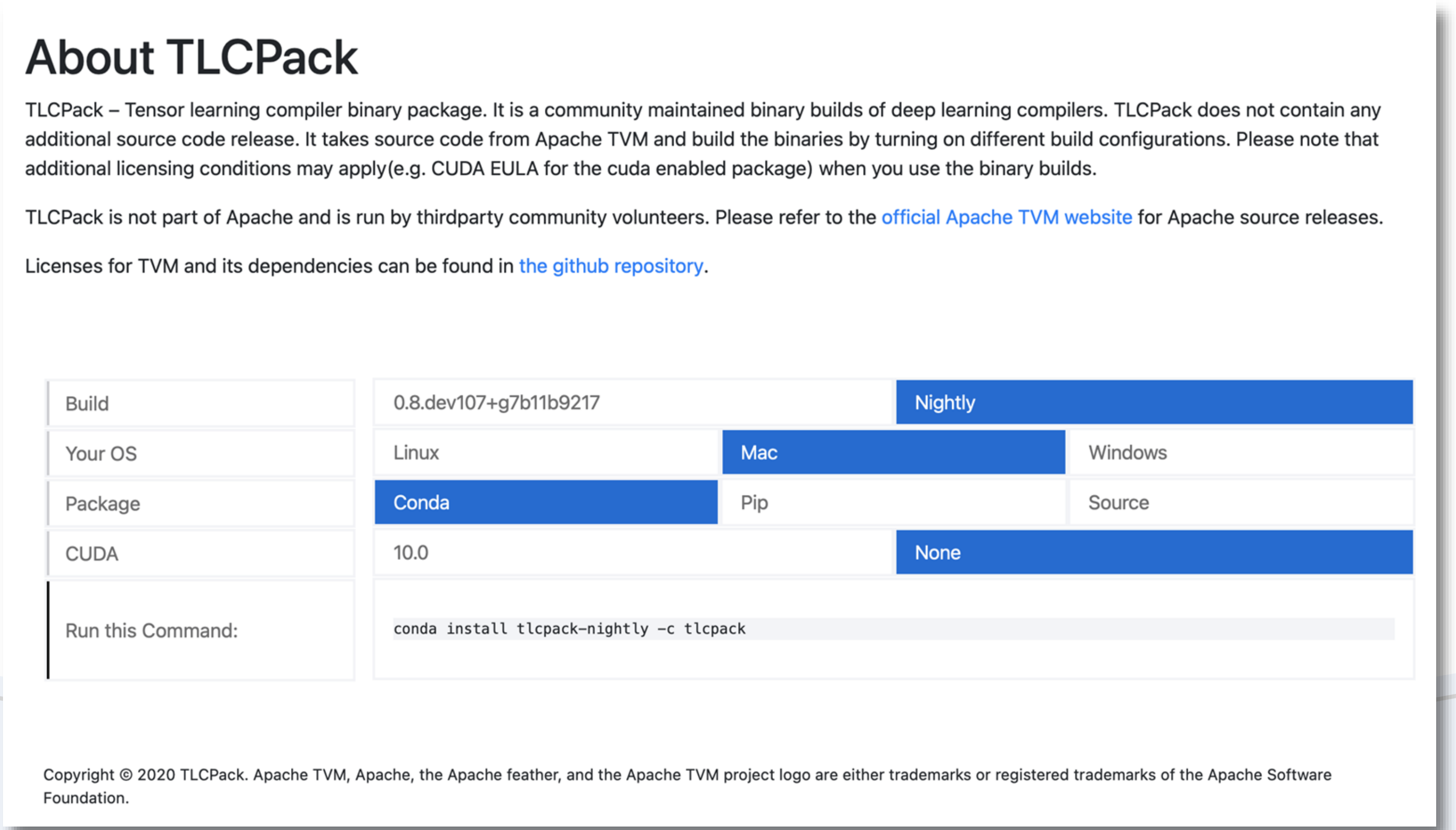
Why should you use TVM?

How does TVM work?

How do you install TVM?

How do you use TVMC?

Install TVM the Easy Way - tlcpack.ai



About TLCPack

TLCPack – Tensor learning compiler binary package. It is a community maintained binary builds of deep learning compilers. TLCPack does not contain any additional source code release. It takes source code from Apache TVM and build the binaries by turning on different build configurations. Please note that additional licensing conditions may apply(e.g. CUDA EULA for the cuda enabled package) when you use the binary builds.

TLCPack is not part of Apache and is run by thirdparty community volunteers. Please refer to the official Apache TVM website for Apache source releases.

Licenses for TVM and its dependencies can be found in the github repository.

Build	0.8.dev107+g7b11b9217	[✓ Nightly]	
Your OS	Linux	[✓ Mac]	Windows
Package	[✓ Conda]	Pip	Source
CUDA	10.0	[✓ None]	
Run this Command:	`conda install tlcpack-nightly -c tlcpack`		

Copyright © 2020 TLCPack. Apache TVM, Apache, the Apache feather, and the Apache TVM project logo are either trademarks or registered trademarks of the Apache Software Foundation.

OctoML

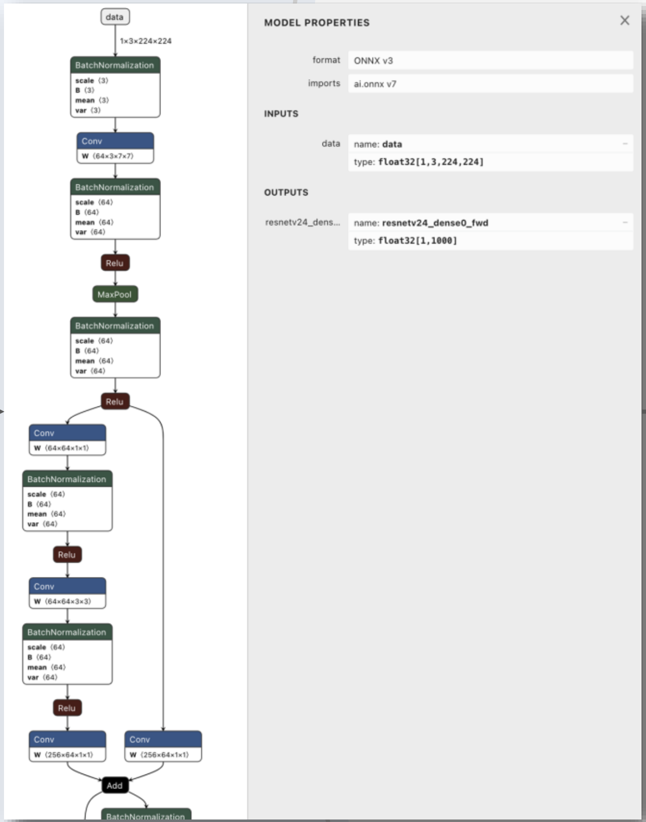
Why should you use TVM?

How does TVM work?

How do you install TVM?

How do you use TVMC?

Model: ONNX representation of ResNet-50v2



MODEL PROPERTIES

format	ONNX v3
imports	ai.onnx v7

INPUTS

data	name: **data**
	type: **float32[1,3,224,224]**

OUTPUTS

resnetv24_dens...	name: **resnetv24_dense0_fwd**
	type: **float32[1,1000]**

```
class='n02123045 tabby, tabby cat'
        with probability=0.610552

class='n02123159 tiger cat'
        with probability=0.367179

class='n02124075 Egyptian cat'
        with probability=0.019365

class='n02129604 tiger, Panthera tigris'
        with probability=0.001273

class='n04040759 radiator'
        with probability=0.000261
```

OctoML

TVMC Demo Roadmap

- Download the ResNet-50v2 Model in ONNX format.
- Download a test image.
- Compile the ONNX model to TVM (Relay).
- Create a tuning record of the model with TVM.
- Compile an optimized model with TVM and compare timings.
- Cross compile the model to run on RPi.
- Remotely tune with RPi.

Download the the ONNX ResNet-50v2 Model

```
resnet50-v2-7.onnx:
        $(info ---===*** Downloading resnet50-v2-7.onnx ***===---)
        curl -L \
https://github.com/onnx/models/raw/master/vision/classification/resnet/model/resnet50-v2-7.onnx \
      -o resnet50-v2-7.onnx

synset.txt:
        $(info ---===*** Downloading additional onnx model file synset.txt ***===---)
        curl -L https://s3.amazonaws.com/onnx-model-zoo/synset.txt -o synset.txt

imagenet-simple-labels.json:
        $(info ---===*** Downloading additional labels for onnx model ***===---)
        curl -L \
https://raw.githubusercontent.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json \
        -o imagenet-simple-labels.json
```

Download and Process the Test Input

```
kitten.jpg:
        $(info ---===*** Downloading kitten.jpg ***===---)
        curl -L https://s3.amazonaws.com/model-server/inputs/kitten.jpg -o kitten.jpg

kitten.npz: kitten.jpg
        $(info ---===*** Converting kitten.jpg to kitten.npz ***===---)
        python3 process_input.py
```

OctoML

Compile and Benchmark the TVM Model

```
resnet50-v2-7.tvm: resnet50-v2-7.onnx
        $(info ---===*** Compiling unoptimized TVM model for resnet50-v2-7 ***===---)
        tvmc compile \
        --target "llvm -mcpu=skylake" \
        --output resnet50-v2-7.tvm \
        resnet50-v2-7.onnx 2> /dev/null

tvm.txt: kitten.npz synset.txt resnet50-v2-7.tvm
        $(info ---===*** Benchmarking unoptimized TVM model for resnet50-v2-7 ***===---)
        echo "=== Begin tvm benchmark results ===" > tvm.txt
        tvmc run \
        --inputs kitten.npz \
        --output predictions.npz \
        --print-time \
        --repeat 5 \
        resnet50-v2-7.tvm > tvm.txt
        python3 process_output.py >> tvm.txt
        echo "=== End tvm benchmark results ===" >> tvm.txt
        echo tvm.txt
```

OctoML

Unoptimized TVM Runtime Results

```
Execution time summary:
 mean (s)   max (s)    min (s)    std (s)
 0.10221    0.08514    0.06656    0.00680

class='n02123045 tabby, tabby cat' with probability=0.610552
class='n02123159 tiger cat' with probability=0.367179
class='n02124075 Egyptian cat' with probability=0.019365
class='n02129604 tiger, Panthera tigris' with probability=0.001273
class='n04040759 radiator' with probability=0.000261
=== End tvm benchmark results ===
```

Tune the Model

```
autotuner_records.json: resnet50-v2-7.onnx
        $(info ---===*** TVM autotuning model for resnet50-v2-7 ***===---)
        tvmc tune \
        --target "llvm -mcpu=skylake" \
        --output autotuner_records.json \
        resnet50-v2-7.onnx 2> /dev/null
```

```
[Task  1/24]  Current/Best:    7.34/  22.55 GFLOPS | Progress: (192/1000) | 346.60 s Done.
[Task  2/24]  Current/Best:   35.23/ 327.39 GFLOPS | Progress: (960/1000) | 919.97 s Done.
[Task  3/24]  Current/Best:   21.31/ 257.22 GFLOPS | Progress: (800/1000) | 626.21 s Done.
[Task  4/24]  Current/Best:   21.06/ 262.27 GFLOPS | Progress: (960/1000) | 524.24 s Done.
[Task  5/24]  Current/Best:   40.97/ 225.42 GFLOPS | Progress: (800/1000) | 626.83 s Done.
[Task  6/24]  Current/Best:   54.34/ 332.29 GFLOPS | Progress: (1000/1000) | 547.70 s Done.
[Task  7/24]  Current/Best:   36.05/ 368.79 GFLOPS | Progress: (1000/1000) | 681.76 s Done.
[Task  8/24]  Current/Best:   25.19/ 326.91 GFLOPS | Progress: (972/1000) | 488.87 s Done.
[Task  9/24]  Current/Best:   14.40/ 334.40 GFLOPS | Progress: (1000/1000) | 443.79 s Done.
[Task 10/24]  Current/Best:   82.75/ 294.44 GFLOPS | Progress: (972/1000) | 538.51 s Done.
[Task 11/24]  Current/Best:   28.24/ 302.67 GFLOPS | Progress: (1000/1000) | 509.88 s Done.
[Task 12/24]  Current/Best:   63.72/ 390.99 GFLOPS | Progress: (1000/1000) | 608.81 s Done.
[Task 13/24]  Current/Best:   49.65/ 355.64 GFLOPS | Progress: (1000/1000) | 445.91 s Done.
[Task 14/24]  Current/Best:   84.85/ 380.20 GFLOPS | Progress: (1000/1000) | 464.45 s Done.
[Task 15/24]  Current/Best:   32.42/ 330.44 GFLOPS | Progress: (1000/1000) | 502.79 s Done.
[Task 16/24]  Current/Best:   60.96/ 345.70 GFLOPS | Progress: (1000/1000) | 629.26 s Done.
[Task 17/24]  Current/Best:   59.07/ 343.07 GFLOPS | Progress: (1000/1000) | 606.57 s Done.
[Task 18/24]  Current/Best:   51.77/ 366.14 GFLOPS | Progress: (980/1000) | 461.96 s Done.
[Task 19/24]  Current/Best:   25.12/ 320.29 GFLOPS | Progress: (1000/1000) | 536.89 s Done.
[Task 20/24]  Current/Best:   33.15/ 398.32 GFLOPS | Progress: (980/1000) | 439.43 s Done.
[Task 21/24]  Current/Best:   47.83/ 394.72 GFLOPS | Progress: (308/1000) | 202.34 s Done.
[Task 22/24]  Current/Best:    9.60/ 331.94 GFLOPS | Progress: (1000/1000) | 696.29 s Done.
[Task 23/24]  Current/Best:   40.74/ 305.27 GFLOPS | Progress: (1000/1000) | 743.44 s Done.
[Task 24/24]  Current/Best:   43.27/ 244.60 GFLOPS | Progress: (1000/1000) | 1053.60 s Done.
```

Compile and Benchmark the Optimized TVM Model

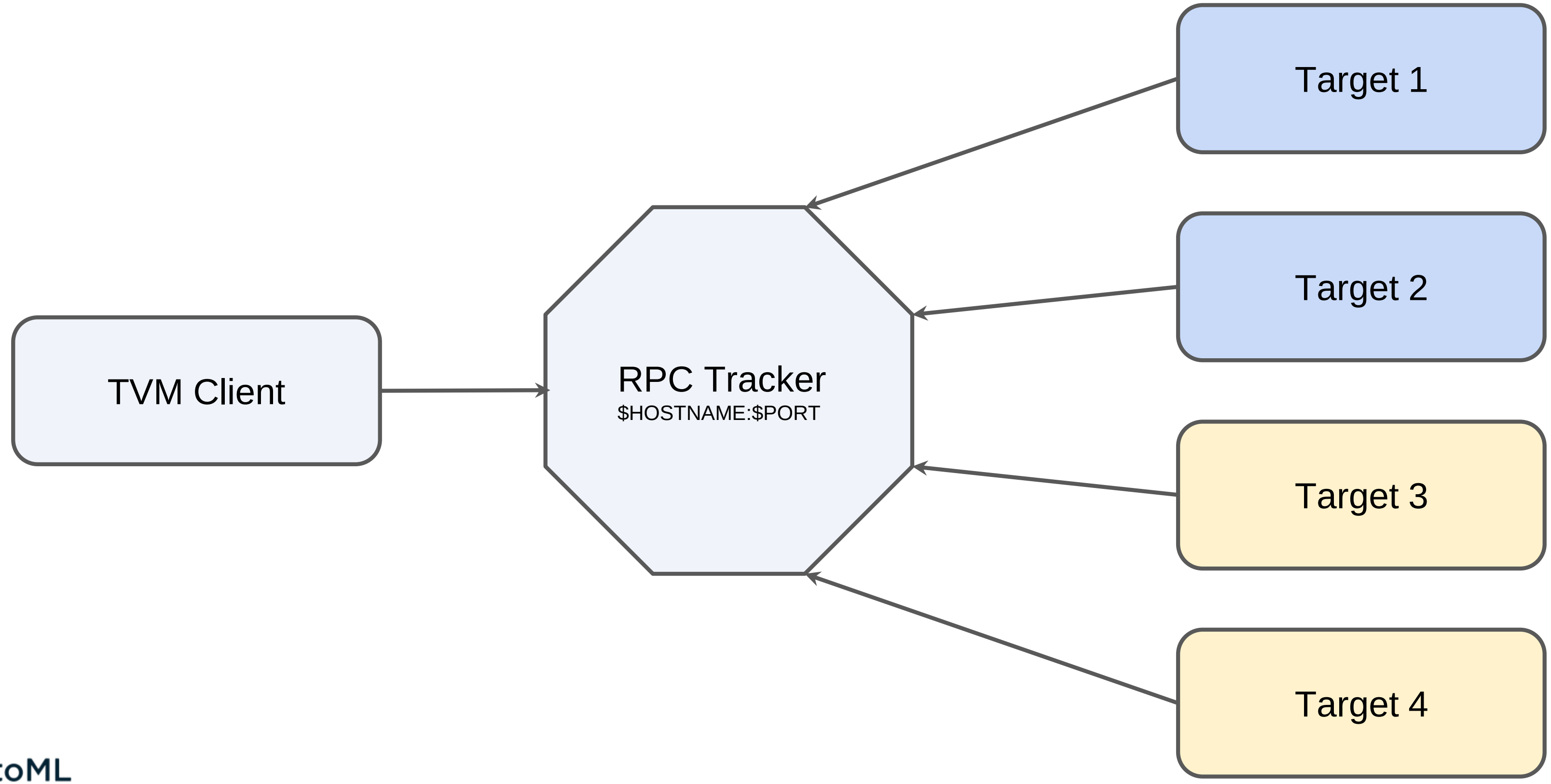
```
resnet50-v2-7-autotuned.tvm: resnet50-v2-7.onnx
	$(info ---===*** Compiling optimized TVM model for resnet50-v2-7 from cache ***===---)
	tvmc compile \
	--target "llvm -mcpu=skylake" \
	--tuning-records autotuner_records.json  \
	--output resnet50-v2-7-autotuned.tvm \
	resnet50-v2-7.onnx 2> /dev/null
tvm-autotuned.txt: kitten.npz synset.txt resnet50-v2-7-autotuned.tvm
	$(info ---===*** Benchmarking optimized TVM model for resnet50-v2-7 ***===---)
	echo "=== Begin tvm-autotuned benchmark results ===" >> tvm-autotuned.txt
	tvmc run \
	--inputs kitten.npz \
	--output predictions.npz \
	--print-time \
	--repeat 5 \
	resnet50-v2-7-autotuned.tvm > tvm-autotuned.txt
	python3 process_output.py >> tvm-autotuned.txt
	echo "=== End tvm benchmark results ===" >> tvm-autotuned.txt
	echo tvm-autotuned.txt
```

OctoML

Optimized TVM Runtime Results

```
=== Begin tvm-autotuned benchmark results ===
Execution time summary:
 mean (s)   max (s)    min (s)    std (s)
 0.03243    0.03655    0.03059    0.00106

class='n02123045 tabby, tabby cat' with probability=0.610552
class='n02123159 tiger cat' with probability=0.367179
class='n02124075 Egyptian cat' with probability=0.019365
class='n02129604 tiger, Panthera tigris' with probability=0.001273
class='n04040759 radiator' with probability=0.000261
```

Remote Execution with TVMC and RPC



OctoML

Cross Compile and the Model for RPi

```
resnet50-v2-7-arm.tvm: resnet50-v2-7.onnx
        $(info ---===*** Compiling unoptimized TVM model for ARM ***===---)
        tvmc compile \
        --target "llvm -device=arm_cpu -mtriple=aarch64-linux-gnu" \
        --cross-compiler aarch64-linux-gnu-gcc \
        --output resnet50-v2-7-arm.tvm \
        resnet50-v2-7.onnx
```

OctoML

Run the Model on RPi using Remote Execution

```
tvm-arm.txt: kitten.npz synset.txt resnet50-v2-7-arm.tvm
        $(info ---===*** Benchmarking unoptimized TVM model for resnet50-v2-7 ***===---)
        echo "=== Begin tvm benchmark results ===" > tvm.txt
        tvmc run \
        --inputs kitten.npz \
        --output predictions.npz \
        --print-time \
        --repeat 5 \
        --rpc-tracker $(TRACKER):$(TRACKER_PORT) \
        --rpc-key raspberry \
        resnet50-v2-7-arm.tvm > tvm-arm.txt
        python3 process_output.py >> tvm-arm.txt
        echo "=== End tvm arm benchmark results ===" >> tvm-arm.txt
        echo tvm-arm.txt
```

OctoML

Tune Model for RPi using Remote Execution

```
autotuner_records-arm.json: resnet50-v2-7-arm.tvm
        $(info ---===*** TVM autotuning model for resnet50-v2-7-arm ***===---) \
        tvmc tune \
        --target "llvm -device=arm_cpu -mtriple=aarch64-linux-gnu" \
        --cross-compiler aarch64-linux-gnu-gcc \
        --output autotuner_records-arm.json \
        --rpc-tracker $(TRACKER):$(TRACKER_PORT) \
        --rpc-key raspberry \
        resnet50-v2-7.onnx 2> /dev/null
```

OctoML

Want to learn more?

- Discuss Forum: https://discuss.tvm.apache.org
- Docs: https://tvm.apache.org/docs
- Source: https://github.com/apache/tvm
- Demo Code: https://github.com/hogepodge/tvm-dem
- Monthly Community Meetings
  - Third Thursday at 9:00 AM PT
- Twitter: @ApacheTVM
- **TinyML Tokyo talk, Introduction to uTVM:**
  https://www.youtube.com/watch?v=R1ZSF5X3JOE

Contact me: choge@octoml.ai or @hogepodge on Twitter

TVM Literature

- TVM: An Automated End-to-End Optimizing Compiler for Deep Learning
- Learning to Optimize Tensor Programs
- **Ansor : Generating High-Performance Tensor Programs for Deep Learning**
- **Nimble: Efficiently Compiling Dynamic Neural Networks for Model Inference**

Copyright Notice

This presentation in this publication was presented as a tinyML® Talks webcast. The content reflects the opinion of the author(s) and their respective companies. 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