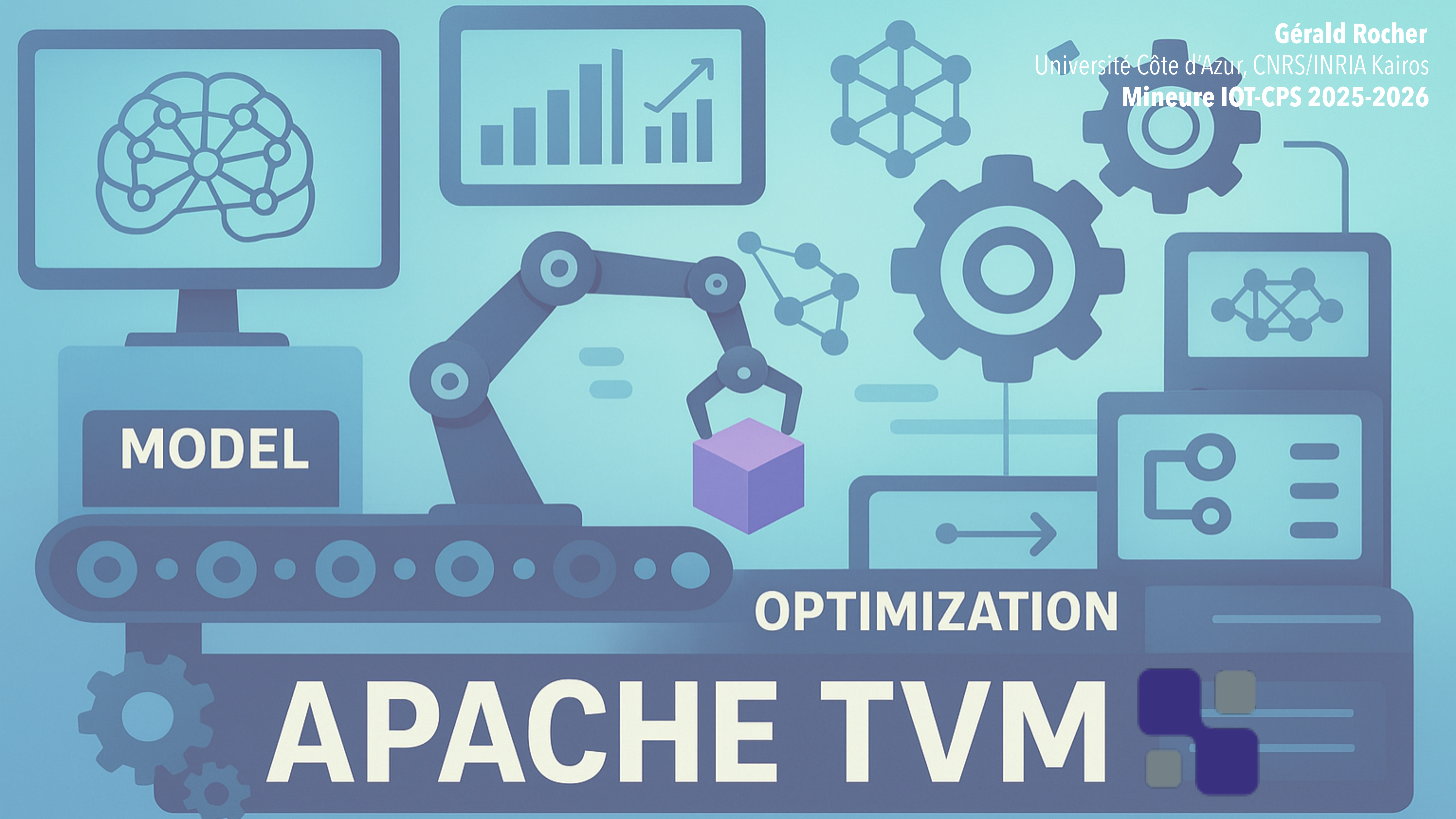
Gérald Rocher
Université Côte d'Azur, CNRS/INRIA Kairos
Mineure IOT-CPS 2025-2026



References

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

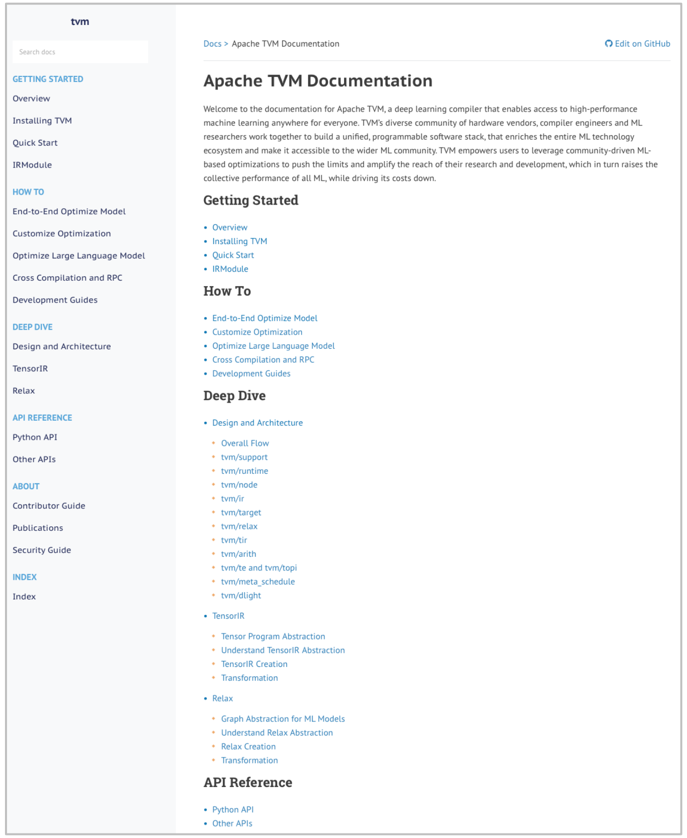
Tianqi Chen and Thierry Moreau, *University of Washington;* Ziheng Jiang, *University of Washington, AWS;* Lianmin Zheng, *Shanghai Jiao Tong University;* Eddie Yan, Haichen Shen, and Meghan Cowan, *University of Washington;* Leyuan Wang, *UC Davis, AWS;* Yuwei Hu, *Cornell;* Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy, *University of Washington*

https://www.usenix.org/conference/osdi18/presentation/chen

This paper is included in the Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18).
October 8–10, 2018 • Carlsbad, CA, USA
ISBN 978-1-939133-08-3

Open access to the Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation is sponsored by USENIX.

Link



tvm

Docs > Apache TVM Documentation — Edit on GitHub

Apache TVM Documentation

Welcome to the documentation for Apache TVM, a deep learning compiler that enables access to high-performance machine learning anywhere for everyone. TVM's diverse community of hardware vendors, compiler engineers and ML researchers work together to build a unified, programmable software stack, that enriches the entire ML technology ecosystem and make it accessible to the wider ML community. TVM empowers users to leverage community-driven ML-based optimizations to push the limits and amplify the reach of their research and development, which in turn raises the collective performance of all ML, while driving its costs down.

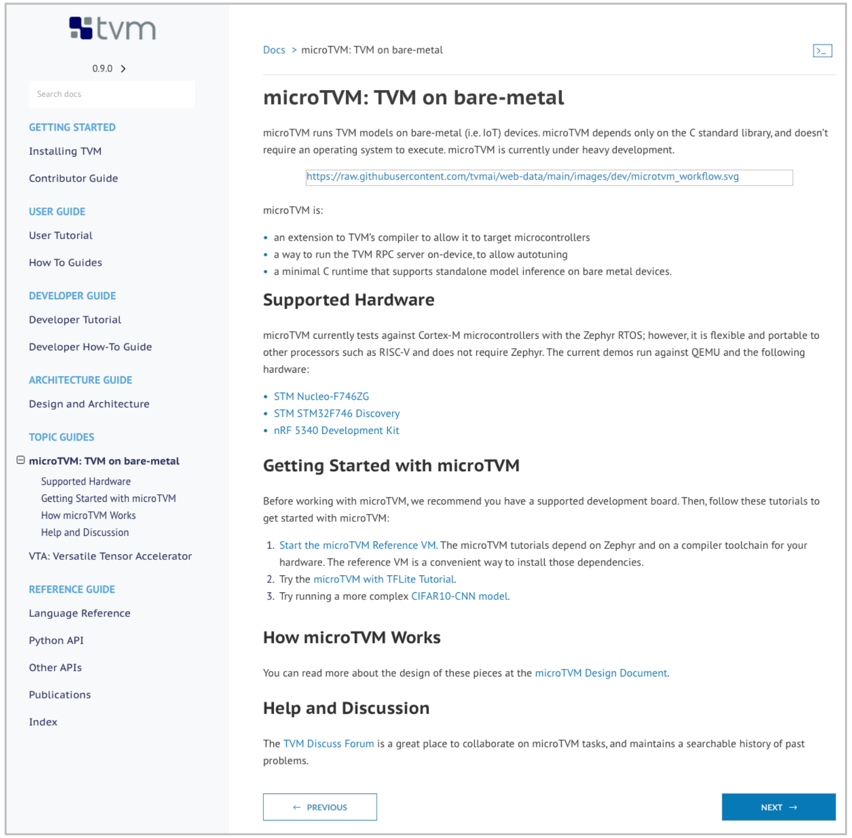
Getting Started
- Overview
- Installing TVM
- Quick Start
- IRModule

How To
- End-to-End Optimize Model
- Customize Optimization
- Optimize Large Language Model
- Cross Compilation and RPC
- Development Guides

Deep Dive
- Design and Architecture
  - Overall Flow
  - tvm/support
  - tvm/runtime
  - tvm/node
  - tvm/ir
  - tvm/target
  - tvm/relax
  - tvm/tir
  - tvm/arith
  - tvm/te and tvm/topi
  - tvm/meta_schedule
  - tvm/dlight
- TensorIR
  - Tensor Program Abstraction
  - Understand TensorIR Abstraction
  - TensorIR Creation
  - Transformation
- Relax
  - Graph Abstraction for ML Models
  - Understand Relax Abstraction
  - Relax Creation
  - Transformation

API Reference
- Python API
- Other APIs

https://tvm.apache.org/docs/



Docs > microTVM: TVM on bare-metal

microTVM: TVM on bare-metal

microTVM runs TVM models on bare-metal (i.e. IoT) devices. microTVM depends only on the C standard library, and doesn't require an operating system to execute. microTVM is currently under heavy development.

microTVM is:
- an extension to TVM's compiler to allow it to target microcontrollers
- a way to run the TVM RPC server on-device, to allow autotuning
- a minimal C runtime that supports standalone model inference on bare metal devices.

Supported Hardware

microTVM currently tests against Cortex-M microcontrollers with the Zephyr RTOS; however, it is flexible and portable to other processors such as RISC-V and does not require Zephyr. The current demos run against QEMU and the following hardware:
- STM Nucleo-F746ZG
- STM STM32F746 Discovery
- nRF 5340 Development Kit

Getting Started with microTVM

Before working with microTVM, we recommend you have a supported development board. Then, follow these tutorials to get started with microTVM:
1. Start the microTVM Reference VM. The microTVM tutorials depend on Zephyr and on a compiler toolchain for your hardware. The reference VM is a convenient way to install those dependencies.
2. Try the microTVM with TFLite Tutorial.
3. Try running a more complex CIFAR10-CNN model.

How microTVM Works

You can read more about the design of these pieces at the microTVM Design Document.

Help and Discussion

The TVM Discuss Forum is a great place to collaborate on microTVM tasks, and maintains a searchable history of past problems.

https://tvm.apache.org/docs/v0.9.0/topic/microtvm/index.html

The slides in this presentation are highly inspired from https://www.youtube.com/watch?v=FDmRVJkCLJ4

https://static.linaro.org/connect/lvc21f/presentations/LVC21F-319.pdf
https://www.edge-ai-vision.com/wp-content/uploads/2021/01/Ceze_2020_Embedded_Vision_Summit_Slides_Final.pdf

What is Apache TVM?

- Many models and ML frameworks landscape; models are mostly interpreted (i.e., not compiled), relying on hand-written libraries targeting specific hardware optimizations,
- Painful and unproductive for users,
- Lacks robustness for application developers whenever the model or hardware change,
- Unsustainable for hardware vendors who often develop their own software stack,
- Explosion of hardware backends (GPUs, TPU, NPUs, ASICs, FPGAs).

Source : https://www.youtube.com/watch?v=FDmRVJkCLJ4

What is Apache TVM?

- **Apache TVM is an open-source automated end-to-end Deep Learning compiler framework for various target devices (Apache TVM website)**
  - TVM targets CPUs, GPUs, NPUs, FPGAs and μControllers (μTVM),
  - Not used for training NNs,
  - Given a trained model, TVM will automatically generate and optimize tensor operators for running the model on the specified target
    - Input models in various formats, like Keras, MXNet, PyTorch, Tensorflow, Tensorflow Lite, CoreML, DarkNet, ONNX, etc.
  - Imlemented in C++ and Python.

Source : https://static.linaro.org/connect/lvc21f/presentations/LVC21F-319.pdf

What is Apache TVM?

Automated, open source, unified optimization and compilation framework for deep learning.

Model In, HW-specific, native code out.

Source : https://www.youtube.com/watch?v=FDmRVJkCLJ4

What is Apache TVM?

TensorFlow mxnet PyTorch

ONNX Keras

NVIDIA intel XILINX

TensorFlow mxnet PyTorch

ONNX Keras

tvm

Automated, open source, unified optimization and compilation framework for deep learning.

Model In, HW-specific, native code out.

NVIDIA intel XILINX

Interpreted

Hand-written kernels

Framework & HW dependent

→

Compiled into native binaries

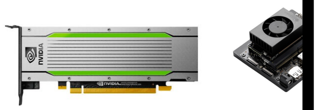
ML-based auto-tuning to generate highly optimized low-level kernels

Framework & HW agnostic

Source : https://www.youtube.com/watch?v=FDmRVJkCLJ4

What is Apache TVM?

TensorFlow mxnet PyTorch
ONNX Keras



NVIDIA

TensorFlow mxnet PyTorch
ONNX Keras

ed optimization
r deep learning.

code out.

Stay tuned! There are many open ML compiler efforts alongside Apache TVM such as MLIR dialects, OpenXLA, Glow, etc.

MLIR OpenXLA GLOW

Interpreted

Hand-written kernels

Framework & HW dependent

Compiled into native binaries

ML-based auto-tuning to generate highly optimized low-level kernels

Framework & HW agnostic

Source : https://www.youtube.com/watch?v=FDmRVJkCLJ4

Apache TVM Stack Overview

TensorFlow mxnet PyTorch ONNX Keras

tvm

ML-based optimizations

Open-source unified foundation for machine learning optimization and compilation.

Relay IR
(High-level data flow graph Intermediate Representation (IR))

TensorIR operators

Pre-optimized op libs
(cuDNN, MKL-DNN, NNPack, ROCm, etc.)

Native Codegen backends
(x86, CUDA, AMD CPU, APU & GPU, ARM CPU M/A class & GPU, FPGAs, etc.)

NVIDIA arm AMD intel XILINX

Goals : enable short time-to-deployment for new models and new HW; easy to extend to new HW and models.

Source : https://www.youtube.com/watch?v=FDmRVJkCLJ4

ML-based optimizations workflow

1. Model Import & Relay IR

- Model ingestion from supported frameworks,
- Conversion into Relay IR.

2. Operator Lowering to TensorIR

- Each operator (e.g., conv2d, matmul) is lowered to TensorIR, a loop-level intermediate representation,
- TensorIR exposes all tunable implementation choices: tiling, loop ordering, vectorization, memory layout, thread mapping, etc.
- This defines the operator-level scheduling search space.

3. ML-based Cost Model

- Instead of exhaustively benchmarking all schedules, TVM trains a cost model to predict performance of candidate schedules on the target hardware,
- Initial measurements seed the model, which is iteratively refined.

4. Auto-Tuning / MetaSchedule Loop

- Candidate TensorIR schedules are generated.
- Predicted runtime is evaluated by the cost model.
- The most promising candidates are executed on real hardware, and results feed back into the cost model.
- Loop: Generate → Predict → Measure → Update.

Source : https://www.youtube.com/watch?v=FDmRVJkCLJ4

ML-based optimizations workflow

5. Best Schedule Selection

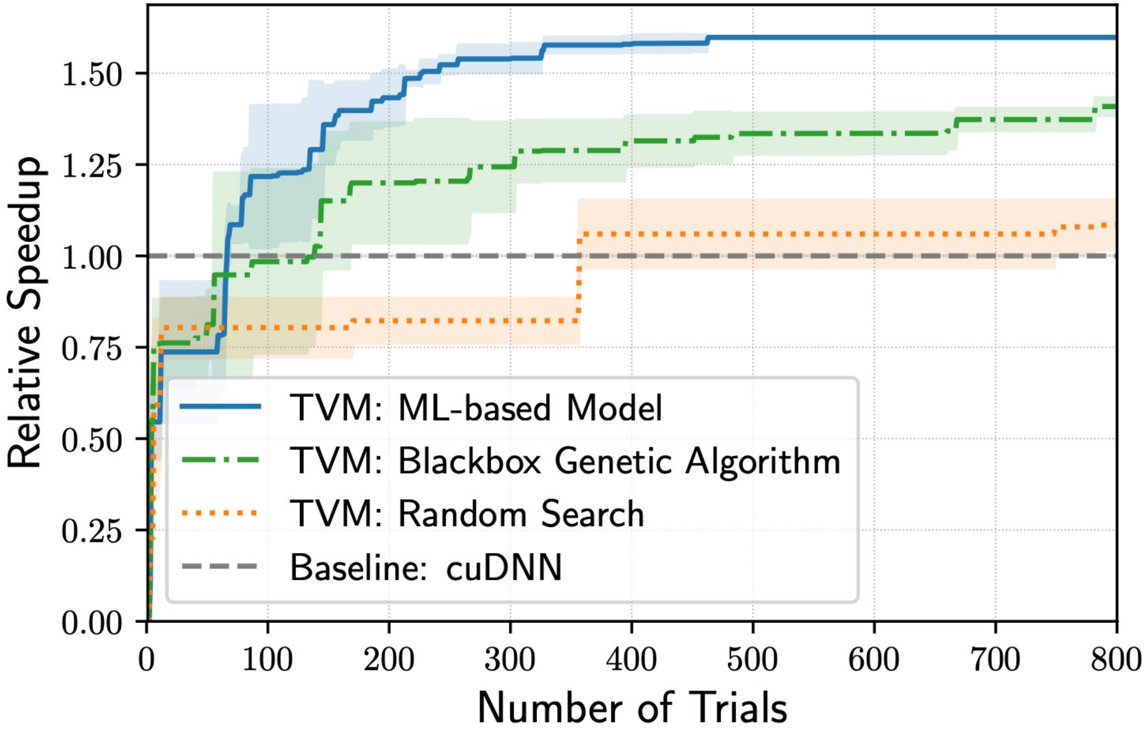
- After enough trials, the fastest schedule per operator is selected.
- Optimized kernels are cached and ready for code generation.

6. Graph-level Optimizations

- Relay-level transformations (operator fusion, quantization, memory reuse) are applied to maximize overall model performance.

7. Code Generation & Deployment

- Final optimized runtime modules are produced (CPU, GPU, or accelerator targets).
- These modules are lightweight and highly tuned for the target hardware.



source

Source : https://www.youtube.com/watch?v=FDmRVJkCLJ4

Example (Python)

```
import tvm
from tvm import relay, auto_scheduler
from tvm.contrib import graph_executor
import onnx
import numpy as np

# Parse ONNX model
onnx_model = onnx.load("model.onnx")

input_name = "input"
input_shape = (1, 3, 224, 224)
input_dtype = "float32"

# Convert ONNX model to Relay IR
# No schedule registered for op XXX → some operators may not have an optimized kernel!
mod, params = relay.frontend.from_onnx(onnx_model, shape={input_name: input_shape}, dtype=input_dtype)

# Set hardware target and tuning options
target = tvm.target.Target("cuda") # ″cuda" or "llvm" for CPU or "rocm" or "metal" or …

# Auto-scheduler search task
tasks, task_weights = auto_scheduler.extract_tasks(mod["main"], params, target)

# Create a measure context
measure_ctx = auto_scheduler.LocalRPCMeasureContext(repeat=1, min_repeat_ms=300)

# Tuning options
tuning_option = auto_scheduler.TuningOptions(
    num_trials=100,
    measure_callbacks=[auto_scheduler.RecordToFile("tuning_records.json")],
    runner=measure_ctx.runner,
)
```

Example (Python)

```python
# Run auto-tuning
for task in tasks:
    print("Tuning task:", task.name)
    tuner = auto_scheduler.TaskScheduler(task, task_weights)
    tuner.tune(tuning_option)

# Compile the model with the best schedule
with auto_scheduler.ApplyHistoryBest("tuning_records.json"):
    lib = relay.build(mod, target=target, params=params)

# Create runtime module and execute inference
dev = tvm.device(str(target), 0)
module = graph_executor.GraphModule(lib["default"](dev))

# Run Inference
input_data = np.random.rand(*input_shape).astype(input_dtype)
module.set_input(input_name, input_data)

module.run()
output = module.get_output(0).asnumpy()
```

Example (Python)

```
# Run auto-tuning
for task in tasks:
    print("Tuning task:", task.name)
    tuner = auto_scheduler.TaskScheduler(task, task_weights)
    tuner.tune(tuning_option)

# Compile t
with auto_s
    lib = re

# Create ru
dev = tvm.d
module = gr

# Run Infer
input_data
module.set_

module.run(
output = mo
```

One can export the compiled model

```
# lib = relay.build(...)
lib.export_library("tuned_model.so")

with open("tuned_model_graph.json", "w") as f:
    f.write(lib.get_graph_json())

with open("tuned_model_params.params", "wb") as f:
    f.write(relay.save_param_dict(lib.get_params()))
```

Example (CLI)

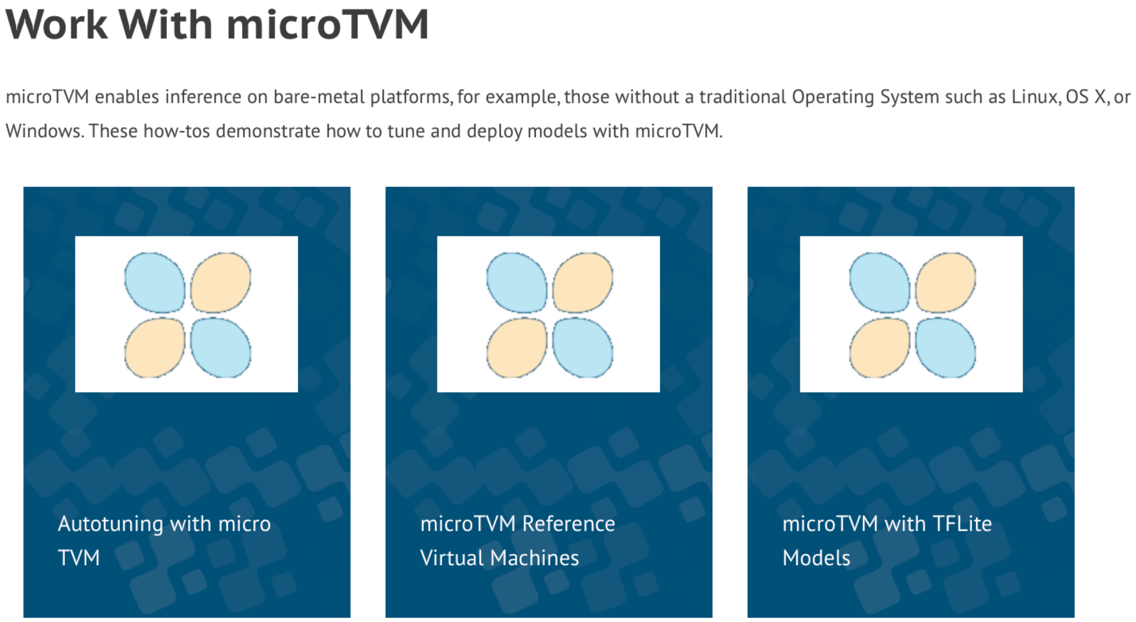
```
# Run auto-tuning
tvmc tune \
    --target "llvm" \
    --output tuning_records.json \
    model.onnx


# Compile the model
tvmc compile \
    --target "llvm" \
    --tuning-records tuning_records.json \
    --output model.tar \
    model.onnx


# Inference
tvmc run \
    --inputs input.npz \
    --output predictions.npz \
    --device cpu \
    --print-time \
    --repeat 100 \
    model.tar
```

μTVM – TVM on Bare-metal (tinyML)

ML Model → μTVM → [Optimized Model | Optimized Operators | Standalone Runtime] → Flash code → Edge Devices



Work With microTVM

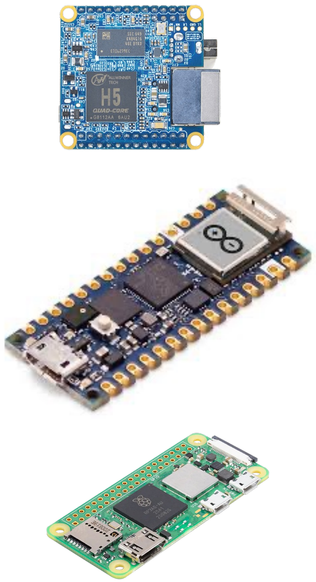
microTVM enables inference on bare-metal platforms, for example, those without a traditional Operating System such as Linux, OS X, or Windows. These how-tos demonstrate how to tune and deploy models with microTVM.

Autotuning with micro TVM

microTVM Reference Virtual Machines

microTVM with TFLite Models

https://tvm.apache.org/docs/v0.8.0/how_to/work_with_microtvm/index.html



Edge Devices

Source: https://www.youtube.com/watch?v=FDmRVJkCLJ4

Apache TVM is an industry standard ML stack

Mobile speech recognition (85x speed-up!)

Unified ML stack for CPU, GPU, NPU built on TVM

Bing query : 112ms (TensorFlow) → 34ms (TVM)

Alexa uses a model optimized with TVM

XILINX AMD Qualcomm …

Source: https://www.youtube.com/watch?v=FDmRVJkCLJ4

Platforms Leveraging Apache TVM

NVIDIA AI Enterprise

- Integrates OctoAI, the former start-up founded by TVM developers.

truefoundry

- A cloud-native ML training and deployment platform built on Kubernetes.
- Facilitates rapid model deployment with scalability and reliability, integrating TVM for optimizations.

Kenning

- An ML framework designed for testing and deploying deep learning applications on edge devices.
- Simplifies the deployment process on edge hardware, utilizing TVM for model optimizations.

EDGE IMPULSE

- An MLOps platform tailored for developing embedded and edge machine learning systems.
- Offers tools for model optimization and deployment on a wide range of hardware targets, incorporating TVM for efficient model execution.