PyTorch CONFERENCE 2025

Bridging PyTorch and TVM: Integrating torch.export.ExportedProgram with Apache TVM

Masahiro Hiramori, Mitsubishi Electric

SCAN ME

Motivation & Problem

- **torch.export** (PyTorch 2.1+) yields a stable single-graph **ExportedProgram**, including **dynamic-shape** semantics and **constraints**; a robust handoff artifact for compilers and runtimes.
- **TVM Relax** is TVM's high-level IR inside an **IRModule**. It handles graph-level transforms and works with **TensorIR** for low-level optimization.

apache/tvm

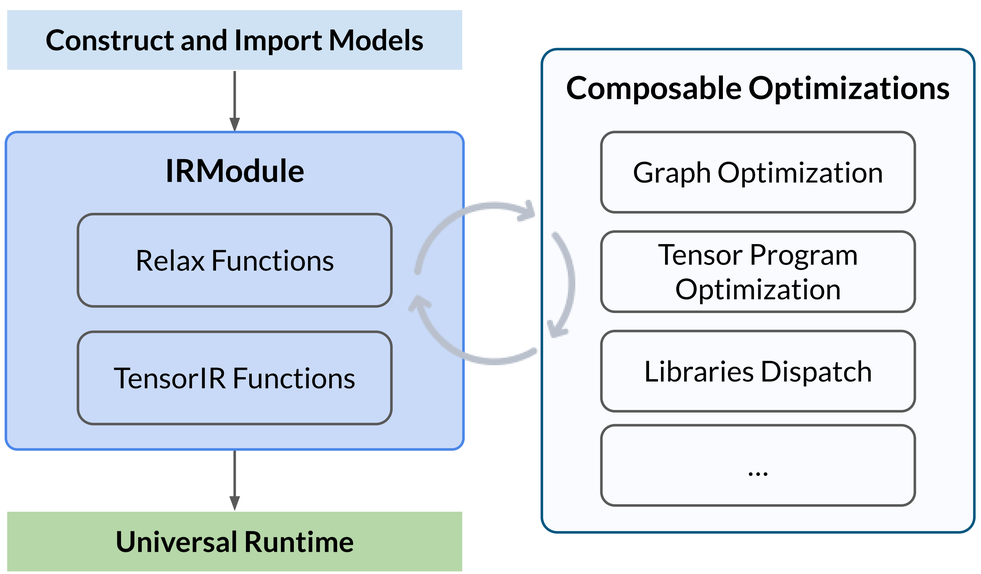
Goal: seamless conversion from `ExportedProgram` → Relax `IRModule` to unlock multi-backend deployment.

Our contribution

We integrated PyTorch's `ExportedProgram` into TVM's Relax frontend to convert models directly into a Relax `IRModule`.

Operator coverage is actively expanding and the importer is usable for many common models today.

Pipeline Overview



Minimal path from ExportedProgram to Relax

1) Export PyTorch Model
2) Import into TVM Relax IRModule
3) Optimize the IRModule
4) Build & Run

```python
# Prepare the model
from torchvision.models.resnet import ResNet18_Weights, resnet18

torch_model = resnet18(weights=ResNet18_Weights.DEFAULT).eval()

# Convert the model to IRModule
import torch
from torch.export import export
from tvm import relax
from tvm.relax.frontend.torch import from_exported_program

with torch.no_grad():
    # Give an example argument to torch.export
    example_args = (torch.randn(1, 3, 224, 224, dtype=torch.float32),)
    exported_program = export(torch_model, example_args)
    mod = from_exported_program(exported_program,
                                keep_params_as_input=True)

mod, params = relax.frontend.detach_params(mod)
```

```python
# IRModule Optimization
import tvm
TOTAL_TRIALS = 20000
target = tvm.target.Target("nvidia/geforce-rtx-4090") # your target device
work_dir = "tuning_logs"
mod = relax.get_pipeline("static_shape_tuning",
                         target=target, total_trials=TOTAL_TRIALS)(mod)

# Build and Deploy
import numpy as np
ex = tvm.compile(mod, target="cuda")
dev = tvm.device("cuda", 0)
vm = relax.VirtualMachine(ex, dev)
## Need to allocate data and params on GPU device
rng = np.random.default_rng()
input_data = rng.random(1, 3, 224, 224).astype("float32")
gpu_data = tvm.runtime.tensor(input_data, dev)
gpu_params = [tvm.runtime.tensor(p, dev) for p in params["main"]]
gpu_out = vm["main"](gpu_data, *gpu_params).numpy()
```

Compatibility & Coverage

- **Over 230 ops are supported**, including unary/binary ops, matrix multiplication, convolutions, activation functions, and tensor transformations.
- **Dynamic shape**: supported through `torch.export` and handled in Relax.
- **Supported TorchVision models**: VGG family; ResNet; ResNeXt; MobileNetV2/V3; MNASNet; EfficientNet (V1/V2); ViT; Swin-T & SwinV2-T; MaxViT.

Current limitations & roadmap

- **Shape constraints**: not yet supported in our importer; planned.
- **Structured control flow (e.g., `torch.cond`)**: not yet supported in our importer; planned.

Try it / Learn more

- PyTorch torch.export overview / tutorial: https://docs.pytorch.org/docs/stable/export.html
- TVM E2E model optimization tutorial: https://tvm.apache.org/docs/how_to/tutorials/e2e_opt_model.html