An Introduction to Torch-MLIR

Marius Brehler

AMD

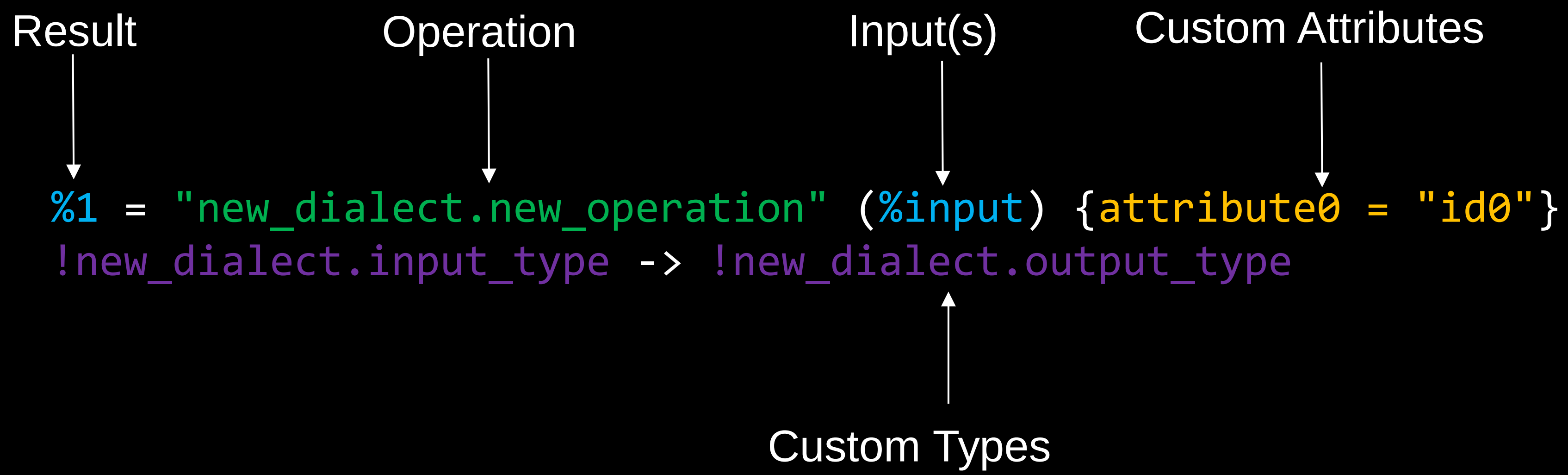
together we advance_

Outline

- What is MLIR?
- What is Torch-MLIR?
- The `torch` Dialect
- Building the Project
- Importing, Converting and Running Models
- How to Get Involved

What is MLIR?

- (Novel) Approach to build a reusable and extensible compiler infrastructure
- Provides an extensible intermediate representation (IR)



Result

Operation

Input(s)

Custom Attributes

```
%1 = "new_dialect.new_operation" (%input) {attribute0 = "id0"}
!new_dialect.input_type -> !new_dialect.output_type
```

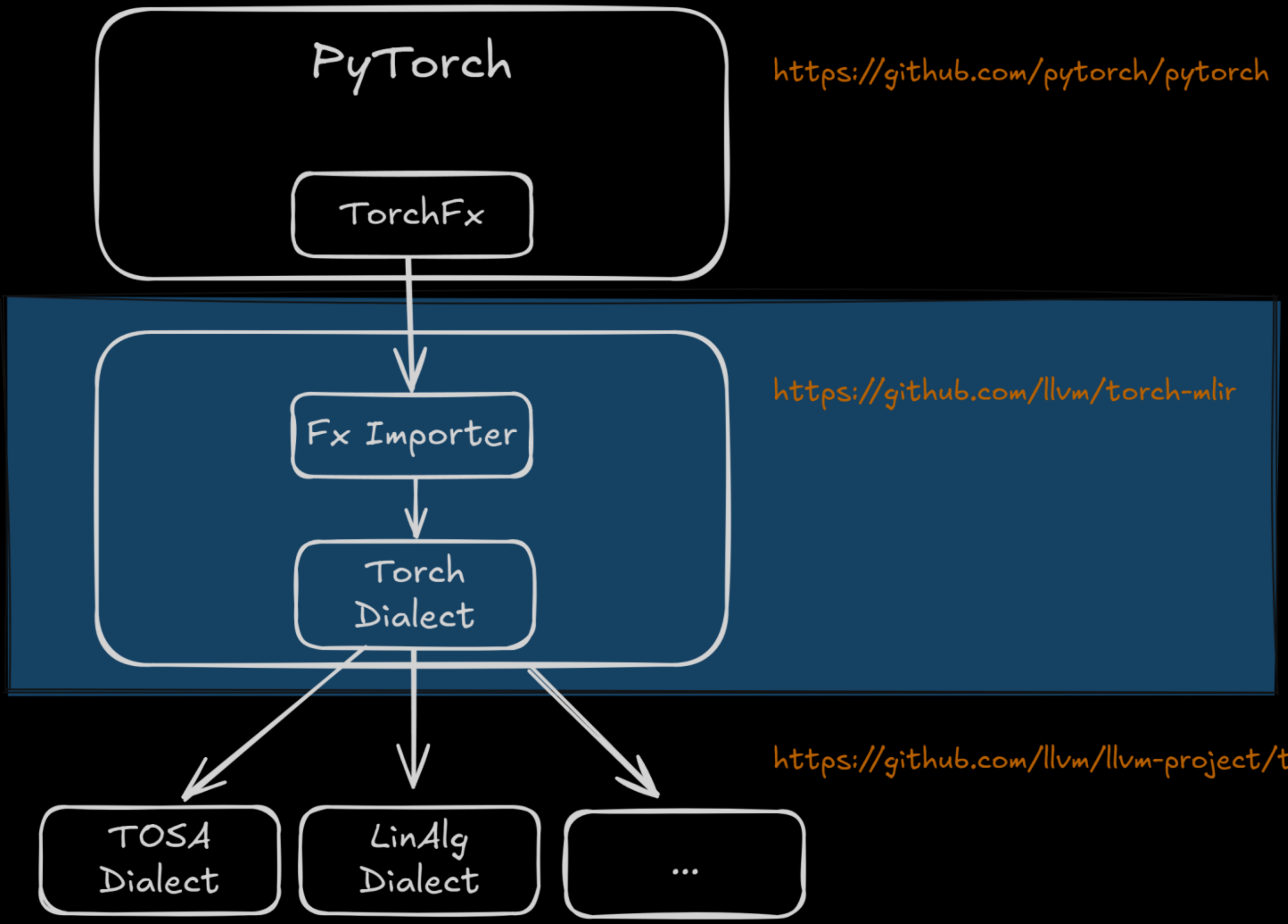
Custom Types

- MLIR is a project of the of the LLVM Foundation

What is Torch-MLIR?

- Aims to provide first class compiler support from the PyTorch ecosystem to the MLIR ecosystem
- Participating in the LLVM Incubator process
- Dual-Licensed under the Apache License 2.0 with LLVM Exceptions and a BSD-style License

What is Torch-MLIR?



https://github.com/pytorch/pytorch

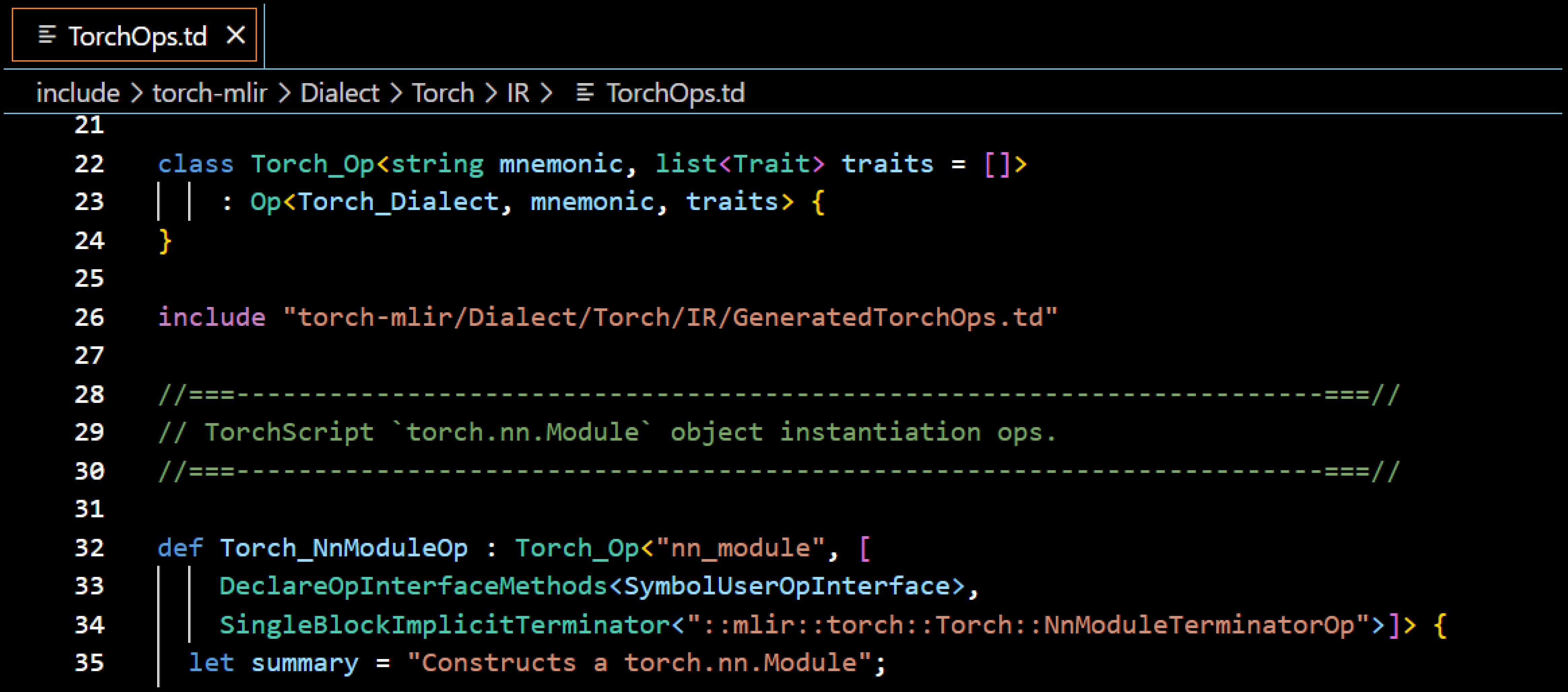
https://github.com/llvm/torch-mlir

https://github.com/llvm/llvm-project/tree/main/mlir

The `torch` Dialect

- The central MLIR abstraction is the `torch` dialect
- Mostly auto-generated based on the PyTorch JIT IR operator registry
- Some manually implemented ops:
  - Used for modeling PyTorch IValue object graphs e.g. `torch.nn_module, torch.class_type`)
  - `torch.global_slot` and related ops to model an incremental lowering of the IValue object graphs
  - Ops supported in the JIT interpreter directly and therefore without a corresponding op in the registry
  - `torch.operator` to represent ops from the registry which haven't been auto-generated

The `torch` Dialect - `TorchOps.td`



```
class Torch_Op<string mnemonic, list<Trait> traits = []>
    : Op<Torch_Dialect, mnemonic, traits> {
}

include "torch-mlir/Dialect/Torch/IR/GeneratedTorchOps.td"

//===----------------------------------------------------------------------===//
// TorchScript `torch.nn.Module` object instantiation ops.
//===----------------------------------------------------------------------===//

def Torch_NnModuleOp : Torch_Op<"nn_module", [
    DeclareOpInterfaceMethods<SymbolUserOpInterface>,
    SingleBlockImplicitTerminator<"::mlir::torch::Torch::NnModuleTerminatorOp">]> {
  let summary = "Constructs a torch.nn.Module";
```

Setup the Project

```
# Clone the project and initialize the submodules

git clone https://github.com/llvm/torch-mlir
cd torch-mlir
git submodule update --init --progress --depth=1



# Create a virtual environment and install dependencies


python3 -m venv mlir_venv
source mlir_venv/bin/activate

python -m pip install --upgrade pip

python -m pip install -r requirements.txt
python -m pip install -r torchvision-requirements.txt
```

Build the Project – Monolithic Build

```
cmake -GNinja -Bbuild-external \
  externals/llvm-project/llvm \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_LINKER_TYPE=LLD \
  -DCMAKE_C_COMPILER_LAUNCHER=ccache -DCMAKE_CXX_COMPILER_LAUNCHER=ccache \
  -DLLVM_ENABLE_ASSERTIONS=ON -DPython3_FIND_VIRTUALENV=ONLY \
  -DLLVM_ENABLE_PROJECTS=mlir \
  -DMLIR_ENABLE_BINDINGS_PYTHON=ON -DLLVM_TARGETS_TO_BUILD=host \
  -DLLVM_EXTERNAL_PROJECTS="torch-mlir" \
  -DLLVM_EXTERNAL_TORCH_MLIR_SOURCE_DIR="$PWD" \
  -DLIBTORCH_CACHE=ON -DLIBTORCH_SRC_BUILD=ON -DLIBTORCH_VARIANT=shared \
  -DTORCH_MLIR_ENABLE_PYTORCH_EXTENSIONS=ON \
  -DTORCH_MLIR_ENABLE_JIT_IR_IMPORTER=ON

cmake --build build-external/ --target check-torch-mlir
```

Build the Project – Component Build (1)

```
# Build and install MLIR
export LLVM_INSTALL_DIR="`realpath install-llvm/`"

cmake -GNinja –Bbuild-llvm \
  externals/llvm-project/llvm \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_LINKER_TYPE=LLD \
  -DCMAKE_C_COMPILER_LAUNCHER=ccache -DCMAKE_CXX_COMPILER_LAUNCHER=ccache \
  -DLLVM_ENABLE_ASSERTIONS=ON -DPython3_FIND_VIRTUALENV=ONLY \
  -DLLVM_ENABLE_PROJECTS=mlir \
  -DMLIR_ENABLE_BINDINGS_PYTHON=ON -DLLVM_TARGETS_TO_BUILD=host \
  -DLLVM_INSTALL_UTILS=ON \
  -DCMAKE_INSTALL_PREFIX="${LLVM_INSTALL_DIR}"

cmake --build build-llvm/ --target install
```

Build the Project – Component Build (2)

```
# Build Torch-MLIR
export LLVM_BUILD_DIR="`realpath build-llvm/`"

cmake -GNinja -Bbuild-component \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_LINKER_TYPE=LLD \
  -DCMAKE_C_COMPILER_LAUNCHER=ccache -DCMAKE_CXX_COMPILER_LAUNCHER=ccache \
  -DLLVM_ENABLE_ASSERTIONS=ON -DPython3_FIND_VIRTUALENV=ONLY \
  -DMLIR_DIR="${LLVM_INSTALL_DIR}/lib/cmake/mlir/" \
  -DLLVM_DIR="${LLVM_INSTALL_DIR}/lib/cmake/llvm/" \
  -DLLVM_EXTERNAL_LIT="${LLVM_BUILD_DIR}/bin/llvm-lit"
  -DMLIR_ENABLE_BINDINGS_PYTHON=ON -DLLVM_TARGETS_TO_BUILD=host \
  -DLIBTORCH_CACHE=ON -DLIBTORCH_SRC_BUILD=ON -DLIBTORCH_VARIANT=shared \
  -DTORCH_MLIR_ENABLE_PYTORCH_EXTENSIONS=ON \
  -DTORCH_MLIR_ENABLE_JIT_IR_IMPORTER=ON

cmake --build build-external/ --target check-torch-mlir
```

Alternative: Install Python Wheels

```
# Install PyTorch
pip install --index-url https://download.pytorch.org/whl/nightly/cpu --pre \
 torch==2.7.0.dev20250120

# Install ONNX
pip install onnx

# Install Torch-MLIR
pip install --find-links https://github.com/llvm/torch-mlir-
release/releases/expanded_assets/dev-wheels torch-mlir
```

A Simple Torch Model

```python
from pathlib import Path
import torch

class SimpleNN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = torch.nn.Linear(16, 10, bias=False)
        self.linear.weight = torch.nn.Parameter(torch.ones(10, 16))
        self.relu = torch.nn.ReLU()
        self.train(False)

    def forward(self, input):
        return self.relu(self.linear(input))

input = torch.randn(2, 16)
model = SimpleNN()
```

A Simple Torch Model Imported to MLIR

```python
from pathlib import Path
import torch

class SimpleNN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = torch.nn.Linear(16, 10, bias=False)
        self.linear.weight = torch.nn.Parameter(torch.ones(10, 16))
        self.relu = torch.nn.ReLU()
        self.train(False)

    def forward(self, input):
        return self.relu(self.linear(input))

input = torch.randn(2, 16)
model = SimpleNN()

from torch_mlir.fx import export_and_import
mlir_module = export_and_import(model, input, output_type="torch")
```

Importing an ONNX Model

```
# Download an ONNX model
wget https://github.com/onnx/models/raw/main/validated/vision/classification/mnist/model/mnist-12.onnx

# Import the model
torch-mlir-import-onnx mnist-12.onnx --opset-version 17 -o mnist.mlir
```

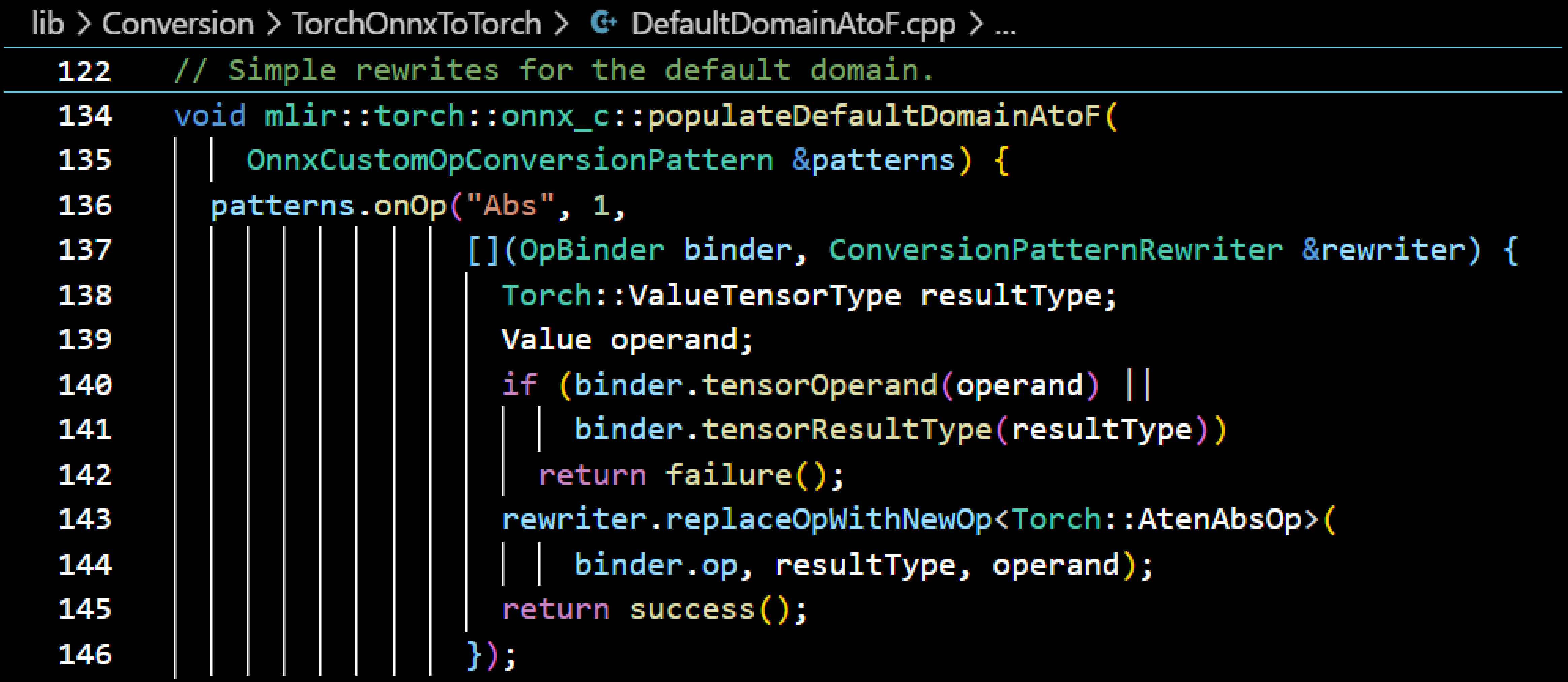
mnist.mlir (Snippet)

```
…

%9 = torch.operator "onnx.Add"(%8, %3) : (!torch.vtensor<[1,8,28,28],f32>,
!torch.vtensor<[8,1,1],f32>) -> !torch.vtensor<[1,8,28,28],f32>
%10 = torch.operator "onnx.Relu"(%9) : (!torch.vtensor<[1,8,28,28],f32>) ->
!torch.vtensor<[1,8,28,28],f32>
%11 = torch.operator "onnx.MaxPool"(%10) {torch.onnx.auto_pad = "NOTSET",
torch.onnx.kernel_shape = [2 : si64, 2 : si64], torch.onnx.pads = [0 : si64, 0
: si64, 0 : si64, 0 : si64], torch.onnx.strides = [2 : si64, 2 : si64]} :
(!torch.vtensor<[1,8,28,28],f32>) -> !torch.vtensor<[1,8,14,14],f32>

…
```

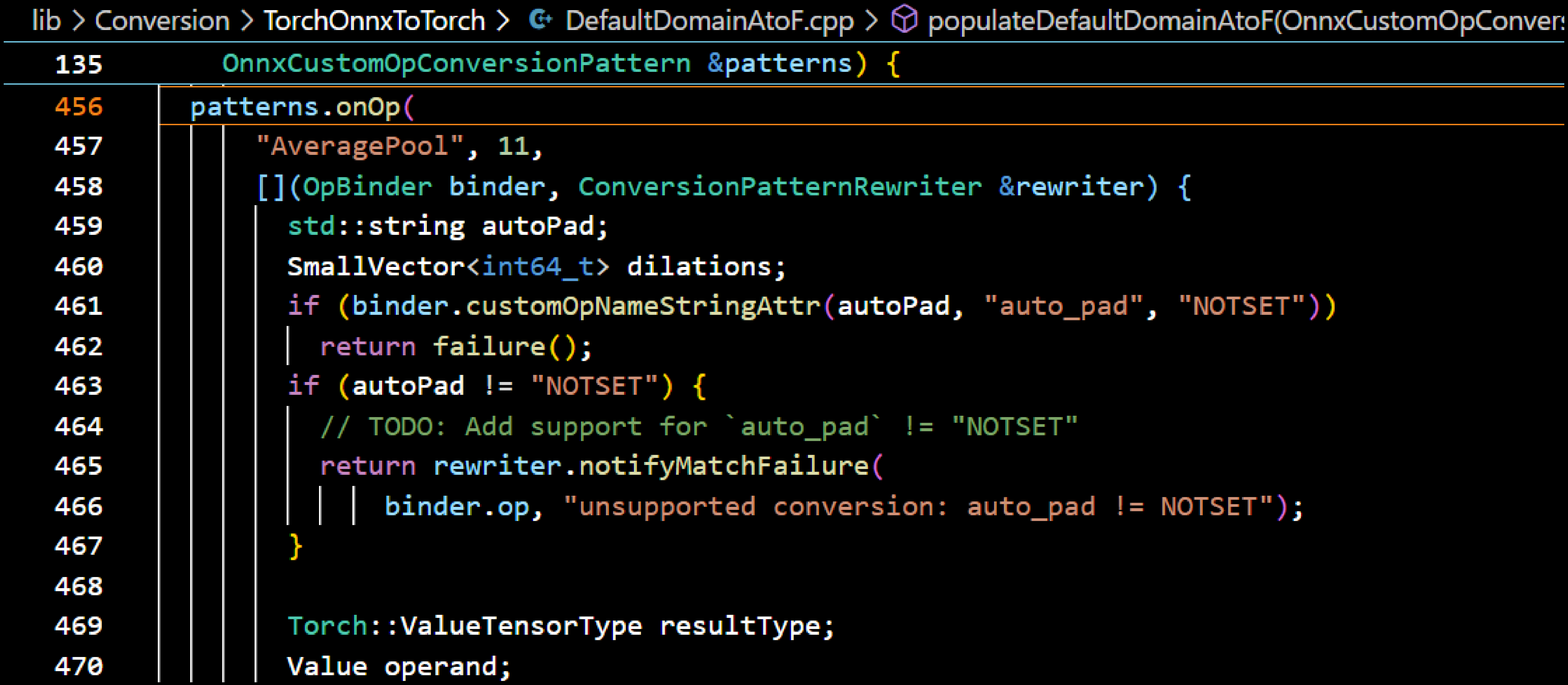
TorchOnnxToTorch - Abs



lib > Conversion > TorchOnnxToTorch > DefaultDomainAtoF.cpp > ...

```
// Simple rewrites for the default domain.
void mlir::torch::onnx_c::populateDefaultDomainAtoF(
    OnnxCustomOpConversionPattern &patterns) {
  patterns.onOp("Abs", 1,
                [](OpBinder binder, ConversionPatternRewriter &rewriter) {
                  Torch::ValueTensorType resultType;
                  Value operand;
                  if (binder.tensorOperand(operand) ||
                      binder.tensorResultType(resultType))
                    return failure();
                  rewriter.replaceOpWithNewOp<Torch::AtenAbsOp>(
                      binder.op, resultType, operand);
                  return success();
                });
```

TorchOnnxToTorch - AveragePool

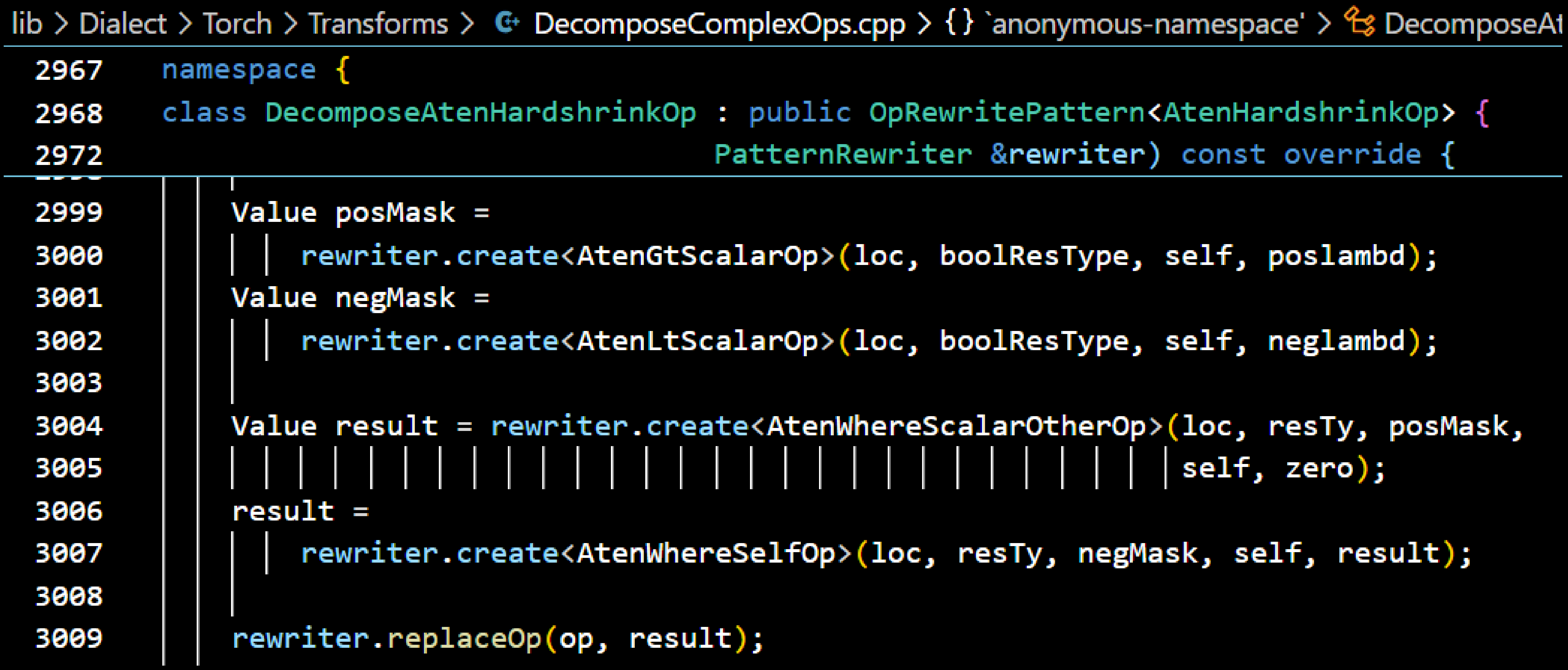


lib > Conversion > TorchOnnxToTorch > DefaultDomainAtoF.cpp > populateDefaultDomainAtoF(OnnxCustomOpConver

```
    OnnxCustomOpConversionPattern &patterns) {
  patterns.onOp(
      "AveragePool", 11,
      [](OpBinder binder, ConversionPatternRewriter &rewriter) {
        std::string autoPad;
        SmallVector<int64_t> dilations;
        if (binder.customOpNameStringAttr(autoPad, "auto_pad", "NOTSET"))
          return failure();
        if (autoPad != "NOTSET") {
          // TODO: Add support for `auto_pad` != "NOTSET"
          return rewriter.notifyMatchFailure(
              binder.op, "unsupported conversion: auto_pad != NOTSET");
        }

        Torch::ValueTensorType resultType;
        Value operand;
```

DecomposeComplexOps



lib > Dialect > Torch > Transforms > DecomposeComplexOps.cpp > {} `anonymous-namespace' > DecomposeAt

```
namespace {
class DecomposeAtenHardshrinkOp : public OpRewritePattern<AtenHardshrinkOp> {
                                PatternRewriter &rewriter) const override {
    Value posMask =
        rewriter.create<AtenGtScalarOp>(loc, boolResType, self, poslambd);
    Value negMask =
        rewriter.create<AtenLtScalarOp>(loc, boolResType, self, neglambd);

    Value result = rewriter.create<AtenWhereScalarOtherOp>(loc, resTy, posMask,
                                                            self, zero);
    result =
        rewriter.create<AtenWhereSelfOp>(loc, resTy, negMask, self, result);

    rewriter.replaceOp(op, result);
```

AMD
together we advance_

Pipelines

- torch-backend-to-linalg-on-tensors-backend-pipeline
- torch-backend-to-stablehlo-backend-pipeline
- torch-backend-to-tosa-backend-pipeline
- torch-function-to-torch-backend-pipeline
- torch-onnx-to-torch-backend-pipeline
- torch-shape-refinement-pipeline
- torch-simplification-pipeline
- torchdynamo-export-to-torch-backend-pipeline
- torchscript-module-to-torch-backend-pipeline

RefBackend

```
# See pt1/examples/fximporter_resnet18.py

from torch_mlir_e2e_test.linalg_on_tensors_backends import refbackend
from torch_mlir_e2e_test.configs.utils import (
    recursively_convert_to_numpy,
)

resnet18 = models.resnet18(pretrained=True).eval()
module = fx.export_and_import(
    resnet18,
    torch.ones(1, 3, 224, 224),
    output_type="linalg-on-tensors",
    func_name=resnet18.__class__.__name__,
)
backend = refbackend.RefBackendLinalgOnTensorsBackend()
compiled = backend.compile(module)
fx_module = backend.load(compiled)
```

How To Get Involved

- Try to import and lower a model by using `torch-mlir-opt`
- Fill issues @ https://github.com/llvm/torch-mlir
- Join discussions @ #torch-mlir (LLVM Discord)
- Implement support for attributes of ONNX operators
- …

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD