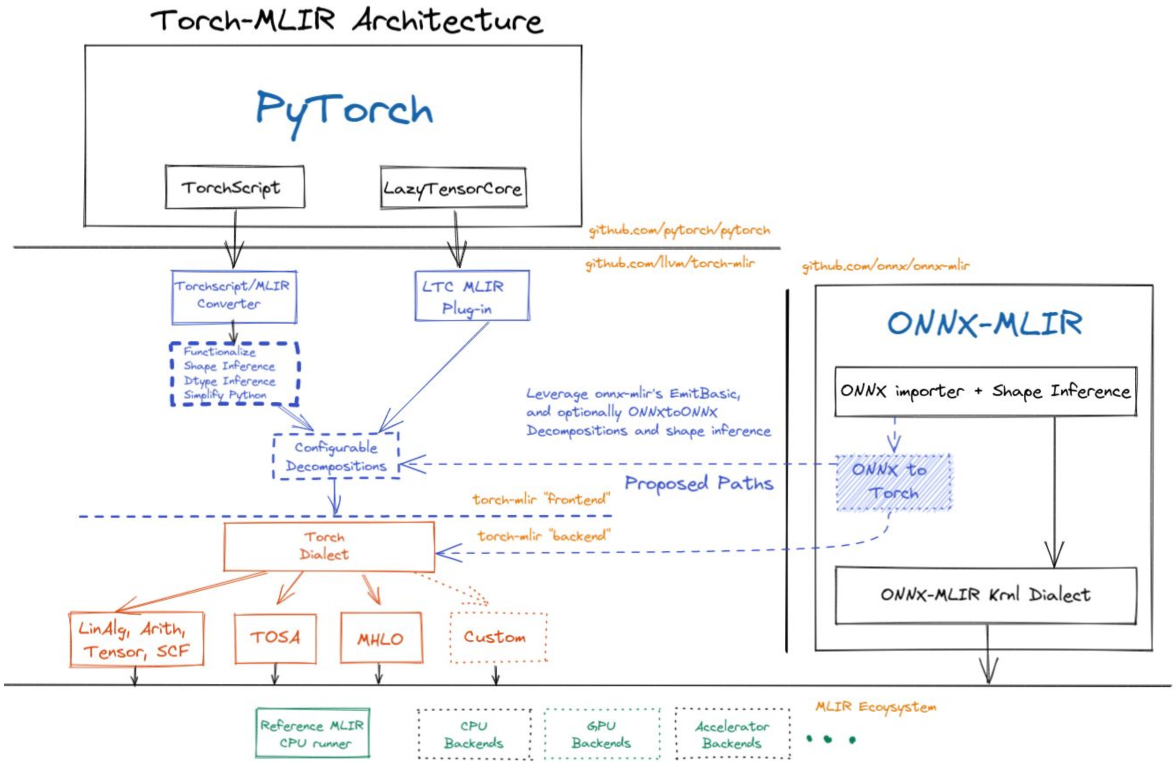
ONNX To Torch Conversion

nod

High Level View



Motivation

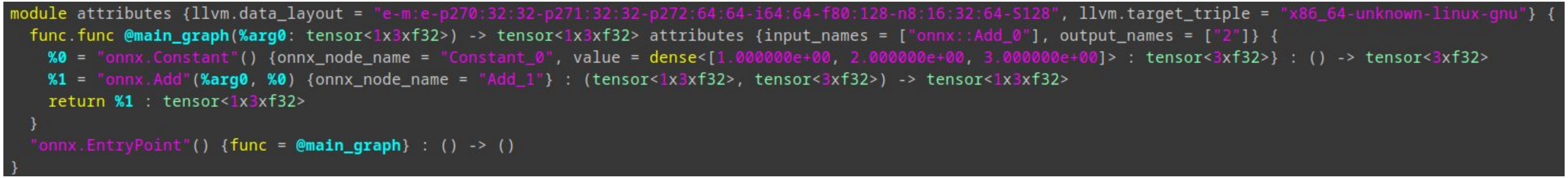
- Unifies efforts between both projects on lowerings to MHLO and TOSA
- Updating to StableHLO only needs to happen in one place
- Gives ONNX-MLIR access to direct lowerings to Linalg
- Nod.ai has multiple customers who use Torch-MLIR that are interested in unified ONNX support

Torch-MLIR Backend Contract

1. All tensors have value semantics
    - Builtin tensors have this for free
2. All tensors have known rank (and ideally as much static shape information as possible)
    - ONNX-MLIR shape inference
    - Torch-MLIR shape inference can optionally be run as well
3. All tensors have known dtype
4. Certain ops require decomposition (e.g. aten._log_softmax)
    - The decompositions in Torch-MLIR are reusable

Passes

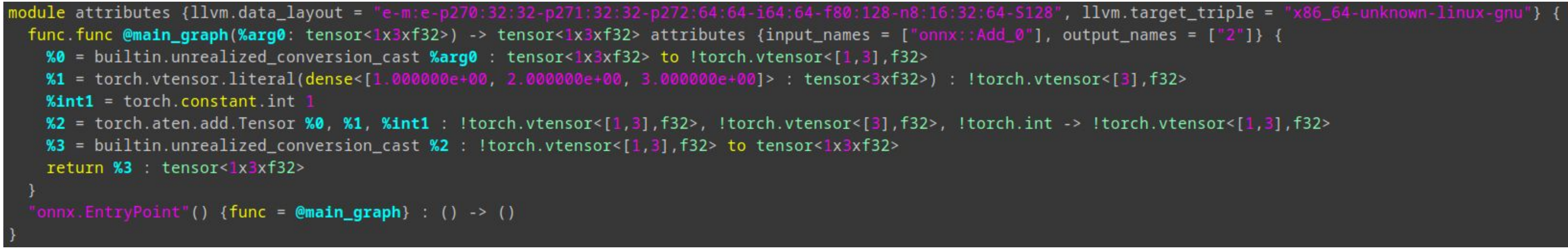
1. “convert-onnx-to-torch”
2. “convert-function-types-to-torch-types”
3. “finalize-torch-type-conversion”
4. “erase-onnx-entry-point”



```
module attributes {llvm.data_layout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128", llvm.target_triple = "x86_64-unknown-linux-gnu"} {
  func.func @main_graph(%arg0: tensor<1x3xf32>) -> tensor<1x3xf32> attributes {input_names = ["onnx::Add_0"], output_names = ["2"]} {
    %0 = "onnx.Constant"() {onnx_node_name = "Constant_0", value = dense<[1.000000e+00, 2.000000e+00, 3.000000e+00]> : tensor<3xf32>} : () -> tensor<3xf32>
    %1 = "onnx.Add"(%arg0, %0) {onnx_node_name = "Add_1"} : (tensor<1x3xf32>, tensor<3xf32>) -> tensor<1x3xf32>
    return %1 : tensor<1x3xf32>
  }
  "onnx.EntryPoint"() {func = @main_graph} : () -> ()
}
```

Passes

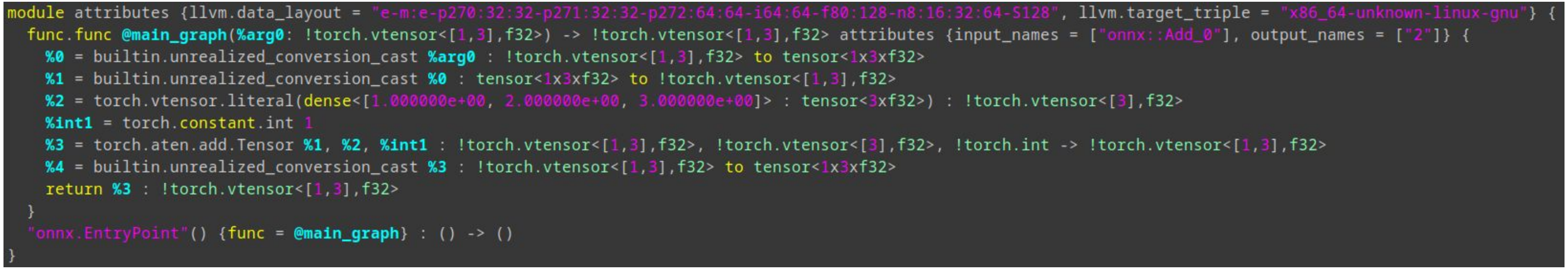
1. **“convert-onnx-to-torch”**
2. “convert-function-types-to-torch-types”
3. “finalize-torch-type-conversion”
4. “erase-onnx-entry-point”



```
module attributes {llvm.data_layout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128", llvm.target_triple = "x86_64-unknown-linux-gnu"} {
  func.func @main_graph(%arg0: tensor<1x3xf32>) -> tensor<1x3xf32> attributes {input_names = ["onnx::Add_0"], output_names = ["2"]} {
    %0 = builtin.unrealized_conversion_cast %arg0 : tensor<1x3xf32> to !torch.vtensor<[1,3],f32>
    %1 = torch.vtensor.literal(dense<[1.000000e+00, 2.000000e+00, 3.000000e+00]> : tensor<3xf32>) : !torch.vtensor<[3],f32>
    %int1 = torch.constant.int 1
    %2 = torch.aten.add.Tensor %0, %1, %int1 : !torch.vtensor<[1,3],f32>, !torch.vtensor<[3],f32>, !torch.int -> !torch.vtensor<[1,3],f32>
    %3 = builtin.unrealized_conversion_cast %2 : !torch.vtensor<[1,3],f32> to tensor<1x3xf32>
    return %3 : tensor<1x3xf32>
  }
  "onnx.EntryPoint"() {func = @main_graph} : () -> ()
}
```

Passes

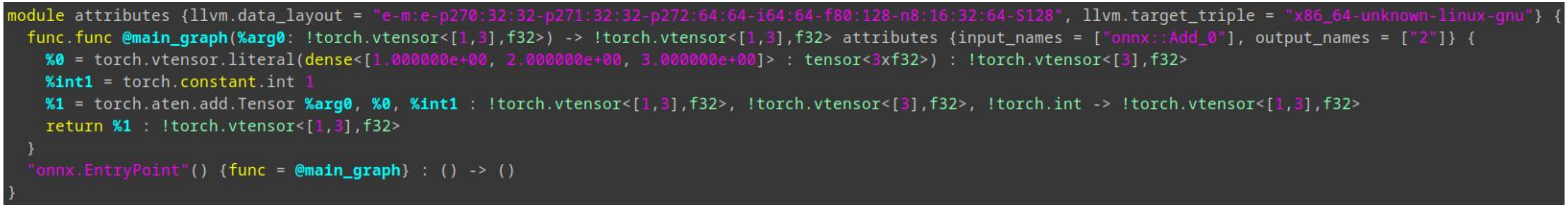
1. “convert-onnx-to-torch”
2. **“convert-function-types-to-torch-types”**
3. “finalize-torch-type-conversion”
4. “erase-onnx-entry-point”



```
module attributes {llvm.data_layout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128", llvm.target_triple = "x86_64-unknown-linux-gnu"} {
  func.func @main_graph(%arg0: !torch.vtensor<[1,3],f32>) -> !torch.vtensor<[1,3],f32> attributes {input_names = ["onnx::Add_0"], output_names = ["2"]} {
    %0 = builtin.unrealized_conversion_cast %arg0 : !torch.vtensor<[1,3],f32> to tensor<1x3xf32>
    %1 = builtin.unrealized_conversion_cast %0 : tensor<1x3xf32> to !torch.vtensor<[1,3],f32>
    %2 = torch.vtensor.literal(dense<[1.000000e+00, 2.000000e+00, 3.000000e+00]> : tensor<3xf32>) : !torch.vtensor<[3],f32>
    %int1 = torch.constant.int 1
    %3 = torch.aten.add.Tensor %1, %2, %int1 : !torch.vtensor<[1,3],f32>, !torch.vtensor<[3],f32>, !torch.int -> !torch.vtensor<[1,3],f32>
    %4 = builtin.unrealized_conversion_cast %3 : !torch.vtensor<[1,3],f32> to tensor<1x3xf32>
    return %3 : !torch.vtensor<[1,3],f32>
  }
  "onnx.EntryPoint"() {func = @main_graph} : () -> ()
}
```

Passes

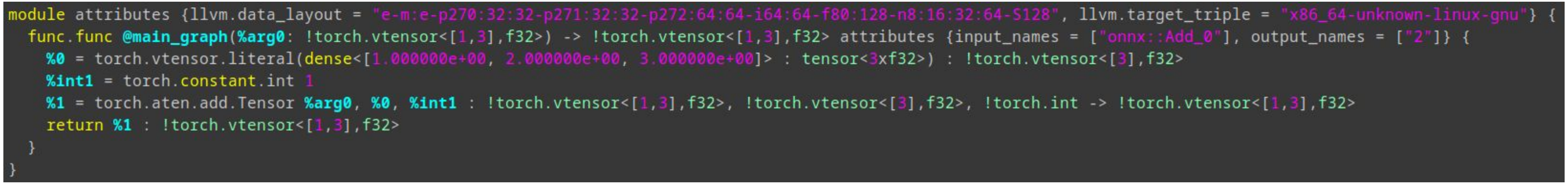
1. “convert-onnx-to-torch”
2. “convert-function-types-to-torch-types”
3. **“finalize-torch-type-conversion”**
4. “erase-onnx-entry-point”



```
module attributes {llvm.data_layout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128", llvm.target_triple = "x86_64-unknown-linux-gnu"} {
  func.func @main_graph(%arg0: !torch.vtensor<[1,3],f32>) -> !torch.vtensor<[1,3],f32> attributes {input_names = ["onnx::Add_0"], output_names = ["2"]} {
    %0 = torch.vtensor.literal(dense<[1.000000e+00, 2.000000e+00, 3.000000e+00]> : tensor<3xf32>) : !torch.vtensor<[3],f32>
    %int1 = torch.constant.int 1
    %1 = torch.aten.add.Tensor %arg0, %0, %int1 : !torch.vtensor<[1,3],f32>, !torch.vtensor<[3],f32>, !torch.int -> !torch.vtensor<[1,3],f32>
    return %1 : !torch.vtensor<[1,3],f32>
  }
  "onnx.EntryPoint"() {func = @main_graph} : () -> ()
}
```

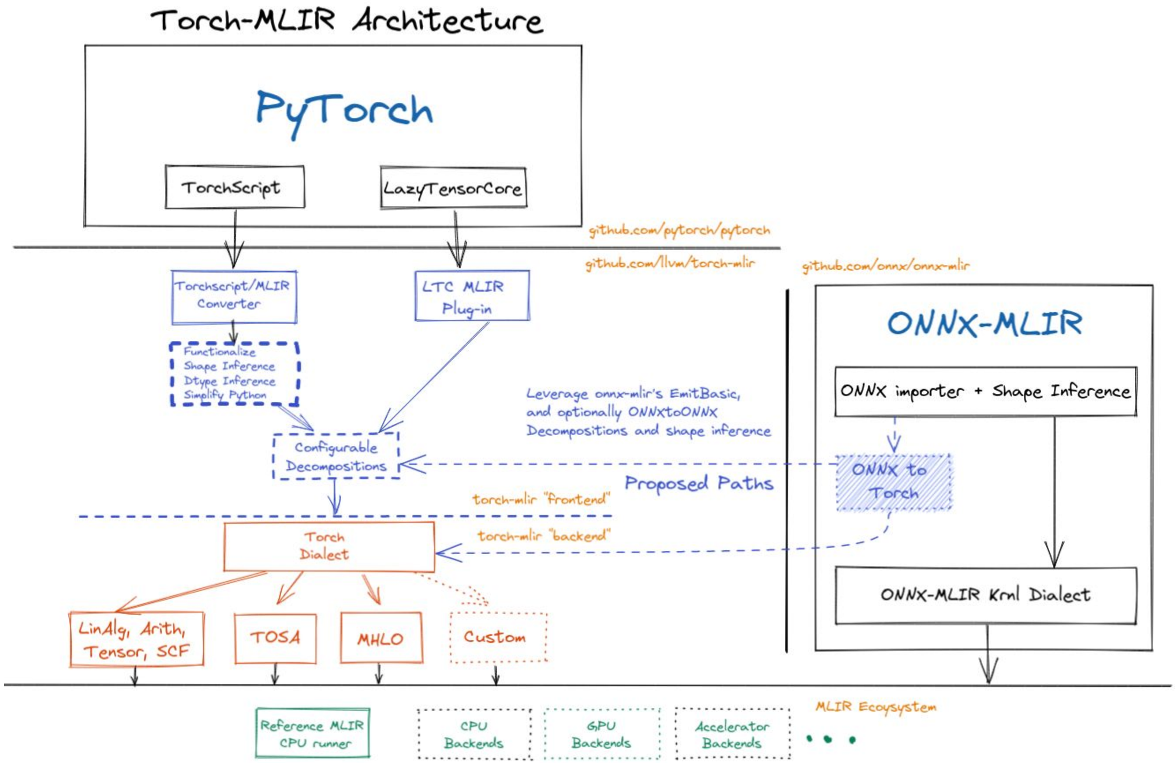
Passes

1. “convert-onnx-to-torch”
2. “convert-function-types-to-torch-types”
3. “finalize-torch-type-conversion”
4. **“erase-onnx-entry-point”**



```
module attributes {llvm.data_layout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128", llvm.target_triple = "x86_64-unknown-linux-gnu"} {
  func.func @main_graph(%arg0: !torch.vtensor<[1,3],f32>) -> !torch.vtensor<[1,3],f32> attributes {input_names = ["onnx::Add_0"], output_names = ["2"]} {
    %0 = torch.vtensor.literal(dense<[1.000000e+00, 2.000000e+00, 3.000000e+00]> : tensor<3xf32>) : !torch.vtensor<[3],f32>
    %int1 = torch.constant.int 1
    %1 = torch.aten.add.Tensor %arg0, %0, %int1 : !torch.vtensor<[1,3],f32>, !torch.vtensor<[3],f32>, !torch.int -> !torch.vtensor<[1,3],f32>
    return %1 : !torch.vtensor<[1,3],f32>
  }
}
```

High Level View


Torch-MLIR Architecture
PyTorch
TorchScript
LazyTensorCore
github.com/pytorch/pytorch
github.com/llvm/torch-mlir
github.com/onnx/onnx-mlir
Torchscript/MLIR Converter
LTC MLIR Plug-in
Functionalize
Shape Inference
Dtype Inference
Simplify Python
Configurable Decompositions
Leverage onnx-mlir's EmitBasic, and optionally ONNXtoONNX Decompositions and shape inference
Proposed Paths
torch-mlir "frontend"
torch-mlir "backend"
Torch Dialect
LinAlg, Arith, Tensor, SCF
TOSA
MHLO
Custom
ONNX-MLIR
ONNX importer + Shape Inference
ONNX to Torch
ONNX-MLIR Krnl Dialect
MLIR Ecoysystem
Reference MLIR CPU runner
CPU Backends
GPU Backends
Accelerator Backends
. . .