Converters SIG Updates

Workshop 04/09/2020

Chin Huang, IBM
Guenther Schmuelling, Microsoft

Converters SIG Updates

- Quick polls… support for onnx-ml and onnx training
- Frontend converters
- Backend converters
- Training user experience discussions
- ONNX-MLIR

Frontend Converter Updates (since Nov)

Pytorch-ONNX exporter

- **ONNX Compliance**
  - Opset 11 fully supported in Pytorch 1.4.0 ; Opset 12 in progress
- **Operators improvement**
  - More than 20+ new operators have been enabled
  - Several existing operators export has been updated
- **New features**
  - Large model export supported in pytorch-onnx exporter
    - HuggingFace GPT-2 Large with 72 layers beyond the 2GB limit exported successfully
  - ONNX checker Integration in pytorch-onnx exporter for high-quality model export
  - Boolean tensor indexing supported in pytorch-onnx exporter
- **More out-of-box model conversion** with Pytorch 1.4
  - Hugging Face BERT/GPT2 models
  - TorchVision models (Opset 11, with fixed input size)
    - Such as FasterRCNN, MaskRCNN, and KeypointRCNN

Frontend Converter Updates (since Nov)

Keras-ONNX converter

- Opset 11 fully supported in Keras2onnx V1.6.0; Opset 12 in progress
- Support tf.keras in tensorflow 2.0/2.1 with subclassing mode
  - Tensorflow 2.2 supported in master branch
- Bidirectional RNN model fully supported
- More out-of-box model conversion
  - tf.keras.applications models; huggingface/transformers.

Sklearn-ONNX converter

- Opset 11 fully supported in skl2onx V1.6.0; Opset 12 in progress
- Supported new models in Scikit-learn 0.22.1
  - StackingRegressor/Classifier, KNNImputer, CategoricalNB, KNeighborsTransformer, HistGradientBoostingRegressor/Classifier, NeighborhoodComponentsAnalysis
- New features added
  - Boolean input type, decision_function and custom parsers supported in skl2onnx

ONNXConverter-Common

- More graph optimizations added in Graph Optimizer
  - MaskRCNN converted from Keras : Observed 50% nodes reduction and up to 7.9x perf speedup with graph optimization

Frontend Converter Updates (since Nov)

Tensorflow-ONNX converter

released version tf2onnx-1.5.6

- supports opset 7-11, tensorflow 1.5-1.14

master

- opset 7-11
- tensorflow 1.11-1.15, tensorflow 2.1-2.2

experimental tensorflow 2 support

- basic models should work
- still some issues with lstm
- uses tensorflow V2 control flow

Backend Converter Updates

- ONNX-TF
  - Tensorflow 2.0 new APIs, tf.function, and persist as SavedModel
  - Opset 11 support almost complete
  - Dynamic/Unknown input shapes
    - ONNX supports it but no standard backend test to verify it
    - ONNX-TF implemented some dynamic shape input test

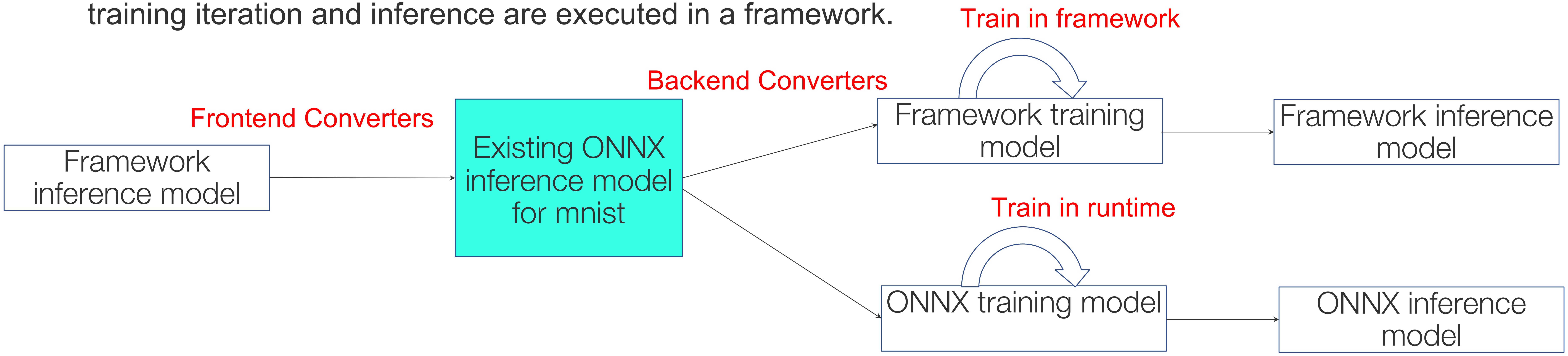
    https://github.com/onnx/onnxtensorflow/blob/master/test/backend/test_dynamic_shape.py
- General
  - ONNX-ML
    - Operators support level (poll)
    - Standard ONNX-ML test cases
  - Training
    - Training support plans (poll)
    - Training user experience in converters

Training User Experience in Converters

Use case #1: The onnx model contains inference only graph and the backend converters/runtimes will generate and run training graph

Expected behaviors:

- Backend (framework converter or runtime) executes training with user inputs or defaults for hyperparameters, loss functions, and optimizers
- No changes to the frontend converters to support this use case
- Runtime persists a ONNX trainable model after n iterations. The next training iteration and inference could be executed in either a runtime or a framework, see use case #2.
- Converter converts to and persists a framework specific trainable model after n iterations. The next training iteration and inference are executed in a framework.

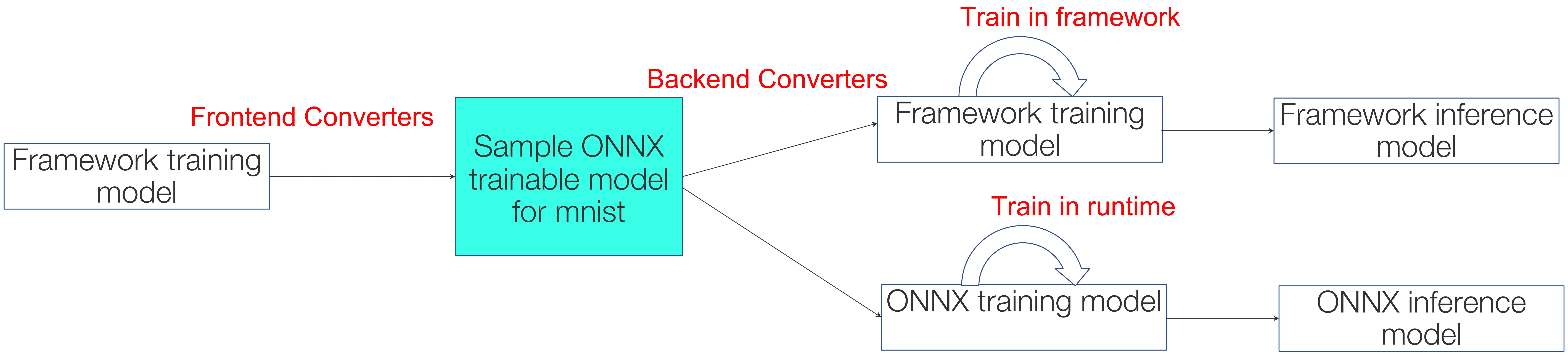


Training User Experience in Converters

Use case #2: The onnx model contains inference and training information

Expected behaviors:

- Frontend converter generates the training info as described in spec, such as hyperparameters, training initialization, algorithm, gradients, loss functions, optimizers
- Backend (framework converter or runtime) executes training as described in the model/training info
- Runtime persists an ONNX trainable model after n iterations. The next training iteration and inference could be executed in either a runtime or a framework.
- Converter converts to and persists a framework specific trainable model after n iterations. The next training iteration and inference are executed in a framework.



Training User Experience in Converters

Training support in Converters

- Currently converters are in various phases of readiness, from no plans, early investigation, to simple prototype

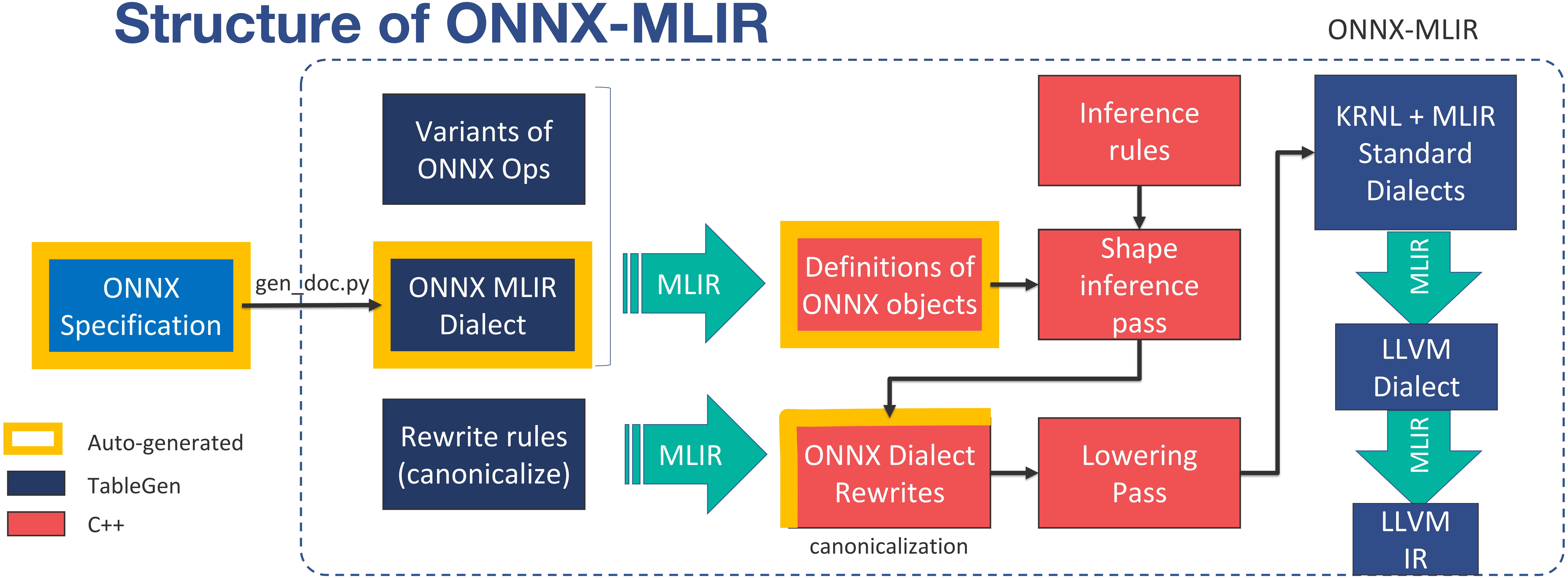
Questions:

- What are the practical (customer) models and scenarios that illustrate training starts from one framework and ends in another (possibly transfer learning)?
- Should a backend framework converter also generate and save an ONNX trainable model in addition to the framework format?
- Any ONNX training APIs, similar to ‘prepare’ for inference, for converters to test and verify training capability?

Implementing ONNX using MLIR

Gheorghe-Teodor (Doru) Bercea
Tong Chen
Alexandre Eichenberger
Tian Jin
Kevin O’Brien

Structure of ONNX-MLIR



- Generate automatically a TableGen description from the Operators.md ONNX specification (gen_doc.py)
- Manually define variants of existing ONNX operations when desired (ex. Conv with no bias: MaxPoolSingleOut).
- Process TableGen files to produce C++ code
- Implement shape inference rules according to ONNX specification and apply them (Example 1).
- Apply rewrite rules to source containing shape-inferred ONNX Dialect operations.
- Lower ONNX Dialect operations to KRNL and Standard Dialects.

Convolution in ONNX dialect

ONNX Model

onnx-mlir --EmitONNXIR mnist.onnx

```
%2 = "onnx.Conv"( %arg0, %arg1) {
            auto_pad = "SAME_UPPER",
            dilations = [1, 1],
            group = 1 : i64,
            kernel_shape = [5, 5],
            strides = [1, 1]
        } : (tensor<1x1x28x28xf32>, tensor<8x1x5x5xf32>) ->
          tensor<1x8x28x28xf32>
```

Defining the ONNX Dialect in TableGen

- Read the ONNX specification and automatically translate specs into an MLIR TableGen file (gen_doc.py).
- TableGen format is later transformed by MLIR TableGen into:

  builders for creating the objects representing the ONNX operation

  getter and setter methods for arguments and attributes (ex. A(), B(), …),

  verification methods

  inference method declarations

  canonicalization methods declarations

```
def ONNXAddOp:ONNX_Op<"Add",
  [NoSideEffect, DeclareOpInterfaceMethods<ShapeInferenceOpInterface>]> {
  let hasCanonicalizer = 1;
  let summary = "ONNX Add operation";
  let description = [{
  "Performs element-wise binary addition. …"
  }];
  let arguments = (ins AnyTypeOf<[AnyMemRef, AnyTensor]>:$A,
                       AnyTypeOf<[AnyMemRef, AnyTensor]>:$B);
  let results = (outs AnyTypeOf<[AnyMemRef, AnyTensor]>:$C);
}
```

Declaring transformations for ONNX Dialect in TableGen

```
%0 = "onnx.MatMul"(%a0, %a1) : (tensor<10x10xf32>, tensor<10x10xf32>) -> tensor<10x10xf32>
%1 = "onnx.Add"(%0, %a2) : (tensor<10x10xf32>, tensor<10x10xf32>) -> tensor<10x10xf32>
```

MLIR

```
Pat<(ONNXAddOp (ONNXMatMulOp:$res $m1, $m2), $m3),
        (ONNXGemmOp $m1, $m2, $m3, (GemmAlpha), (GemmBeta), (GemmTransA), (GemmTransB))>
```

```
%0 = "onnx.Gemm"(%a0, %a1, %a2) {
    alpha = 1.000000e+00 : f32,
    beta = 1.000000e+00 : f32,
    transA = 0 : i64,
    transB = 0 : i64 } : (tensor<10x10xf32>, tensor<10x10xf32>, tensor<10x10xf32>) ->
                          tensor<10x10xf32>
```

Where we are with development.

- Full support for representation of ONNX operations within MLIR framework.
- Growing number of operations can be lowered from ONNX -> MLIR Dialects -> LLVM.
- Support lowering of MNIST from ONNX Dialect to LLVM.

In Progress:

- Laying down some infrastructure that will allow the user to control compiling and running models in general not just MNIST.
- More operation lowering support.
- Explore optimal ways to encode ONNX model metadata -
  - Opset version, initializers, big constants.
- Support operation versioning -
  - ONNX-MLIR can potentially help with converter efforts too.
- More tests!

Thank You!
and
Join our SIG Meetings
https://lists.lfai.foundation/g/onnx-sig-operators/