Compiling ONNX Neural Network Models Using MLIR

Author: Tian Jin, Gheorghe-Teodor Bercea, Tung D. Le, Tong Chen, Gong Su, Haruki Imai, Yasushi Negishi, Anh Leu, Kevin O'Brien, Kiyokuni Kawachiya, Alexandre E. Eichenberger

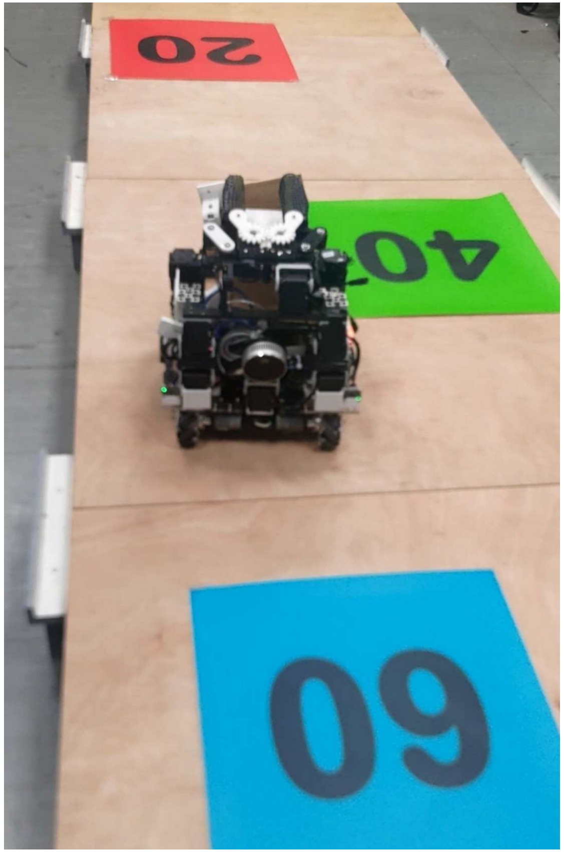
Advisor: 陳鵬升 教授
Presenter: 洪祐鈞

What Does this project do?

- Objective: Transform ONNX graphs into code
- Approach: Utilizes LLVM/MLIR compiler technology
- Focus: Inference
- Goal:
  - Implementation with stand-alone runtime support.
  - Integration to other MLIR compilers.

What is ONNX?

- ONNX (Open Neural Network Exchange)
- A common format for deep learning model.
- Supported by Pytorch, TensorFlow, Caffe,…
  - torch.onnx.export(model, example_input, onnx_path)
- Real life example: TensorRt
  - Deploy onnx model to NVIDIA GPUs.
  - Optimization and acceleration on inference:
    - Kernel auto-tuning.
    - Precision quantization.



What is ONNX?

- LeakyRelu Operation Example:

Listing 1: ONNX model for LeakyRelu operator (printe
using 'protoc' command).

```
ir_version: 3
producer_name: "backend-test"
graph {
  node {
    input: "x"
    output: "y"
    op_type: "LeakyRelu"
    attribute {
      name: "alpha"
      f: 0.1
      type: FLOAT
    }
  }
  name: "test_leakyrelu"
  input {
    name: "x"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
            dim_value: 3
          }
          dim {
            dim_value: 4
          }
          dim {
            dim_value: 5
          }
```

```
  output {
    name: "y"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
            dim_value: 3
          }
          dim {
            dim_value: 4
          }
          dim {
            dim_value: 5
          }
        }
      }
    }
  }
}
opset_import {
  version: 9
}
```

What is ONNX?

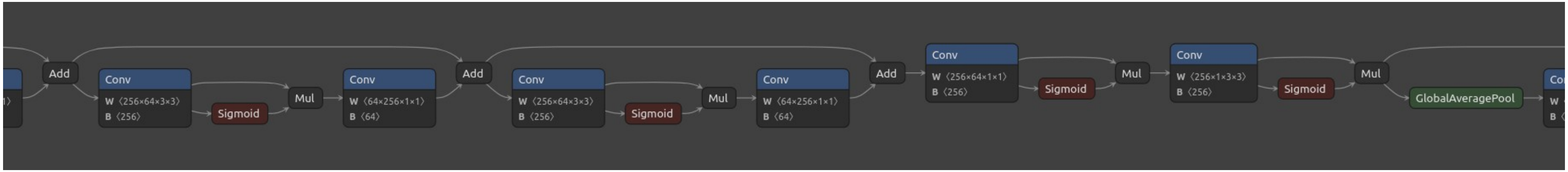
- Conv Operation Example:

```
onnx_model = onnx.load(onnx_file_path)
print(onnx_model)
```

```
ir_version: 8
producer_name: "pytorch"
producer_version: "2.1.1"
graph {
  node {
    input: "input.1"
    input: "onnx::Conv_1396"
    input: "onnx::Conv_1397"
    output: "/features/features.0/features.0.0/Conv_output_0"
    name: "/features/features.0/features.0.0/Conv"
    op_type: "Conv"
    attribute {
      name: "dilations"
      ints: 1
      ints: 1
      type: INTS
    }
    attribute {
      name: "group"
      i: 1
      type: INT
    }
    attribute {
      name: "kernel_shape"
      ints: 3
      ints: 3
      type: INTS
    }
    attribute {
      name: "pads"
      ints: 1
      ints: 1
      ints: 1
      ints: 1
      type: INTS
    }
    attribute {
      name: "strides"
      ints: 2
      ints: 2
      type: INTS
    }
  }
```

What is ONNX?

- EfficientNet v1, visualized by Netron.
- Input, output, dimension, weight, bias, operation,...



What is MLIR?

- MLIR (Multi-Level Intermediate Representation)
- Motivation:
  - Various kind of hardware for domain specific accelerator. (DSA)
    - Historically, there was scalar and static vector types, now MLIR has tensor type.
  - Lowering the development cost of domain specific compiler.
  - Devs usually build their own high-level IR before going to lower level.
- Chris Lattner: “LLVM is a subset of MLIR”

What is MLIR?

- MLIR Operation (Op):
  - Op is central semantic unit of mlir.
    - Instruction, functions, modules,…
  - Encourage users to define custom Ops
  - Ops contains a list of regions.
  - Regions contain list of blocks.
    - In a sense of control flow graphs.
  - Blocks can contain list of ops
    - Allow recursive structures.

```
%results:2 = "d.operation"(%arg0, %arg1) ({
  // Regions belong to Ops and can have multiple blocks.      Region
  ^block(%argument: !d.type):                                 Block
    // Ops have function types (expressing mapping).
    %value = "nested.operation"() ({
      // Ops can contain nested regions.                  Region
      "d.op"() : () -> ()
    }) : () -> (!d.other_type)
    "consume.value"(%value) : (!d.other_type) -> ()
  ^other_block:                                               Block
    "d.terminator"() [^block(%argument : !d.type)] : () -> ()
  })
// Ops can have a list of attributes.
{attribute="value" : !d.type} : () -> (!d.type, !d.other_type)
```

Figure 3: Operation (Op) is a main entity in MLIR; operations contain a list of regions, regions contain a list of blocks, blocks contains a list of Ops, enabling recursive structures

MLIR: A Compiler Infrastructure for the End of Moore's Law

What is MLIR?

- MLIR Dialects:
  - A group of operations into a namespace.
- MLIR Optimization Passes:
  - Can be expressed to DRRs (Declarative rewriting Rules)
    - Tablegen, C++ code
  - Conversion:
    - KrnlToAffine, KrnlToLLVM, ONNXToKrnl, ONNXToTOSA, ONNXToHLO
  - Transformation:
    - Decompose, shape infererence, ConvOpt
  - Translation: Involve external representation.

What is MLIR?

- MLIR ecosystem:
  - Tensorflow: where MLIR originated
  - mhlo: Part of TF, Dynamic scaled XLA (for accelerating linear algerbra)
  - tfrt: TF runtime
  - torch-mlir: connecting pytorch and mlir ecosystem
  - circt: Hardware/Software Co-design
  - iree: Deep Learning E2E compiler. Kinda like TVM

What does the project provide?

- ONNX Dialect that can be integrated to other projects.
- Compiler interfaces:
  - Lowers ONNX graphs to MLIR, LLVM, C/C++, Python and Java.
  - Graph-to-code transformation, stand-alone runtime environment.
- Target:
  - Generic CPUs. (Linux, Windows, macOS)
  - IBM's Telum AI accelerator. (IBM z16)
    - A server that can handle huge amount of queries.

What does the project provide?

- What does it means?
  - Easy to write optimization of CPU and custom accelerators
    - MLIR
  - Easy to deploy
    - No libraries need, only stand-alone driver and runtime support.
  - Integrate ONNX to MLIR eco-system

Motivation:

- Many deep learning frameworks use their own optimized libraries for specific accelerators during inference.
- Drawbacks:
  - Number of supported accelerator of model is limited to libraries.
  - Needed to install various libraries.
  - The implementation varies for the same operation.
- Proposed Solution: Developing a compiler that rewrites a trained model to native code for a target hardware.

Architecture:

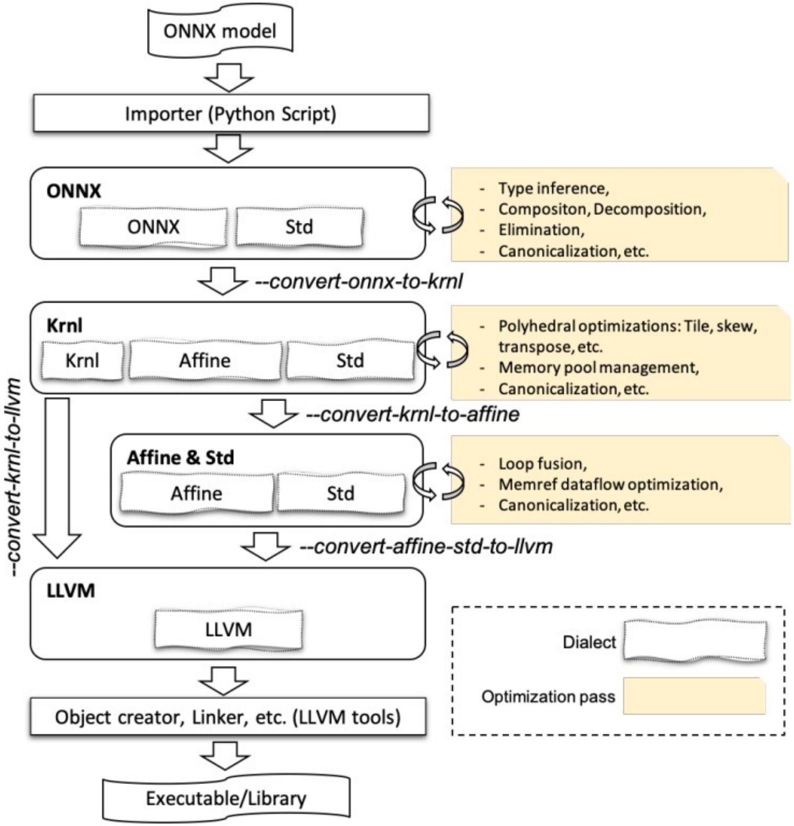


Fig. 2: Architecture of onnx-mlir. Names prefixed with '--' are passes.

Architecture:

- 4 abstraction layer:
- 5 main Dialect:
  - ONNX
  - Krnl
  - Std
  - Affine
  - LLVM

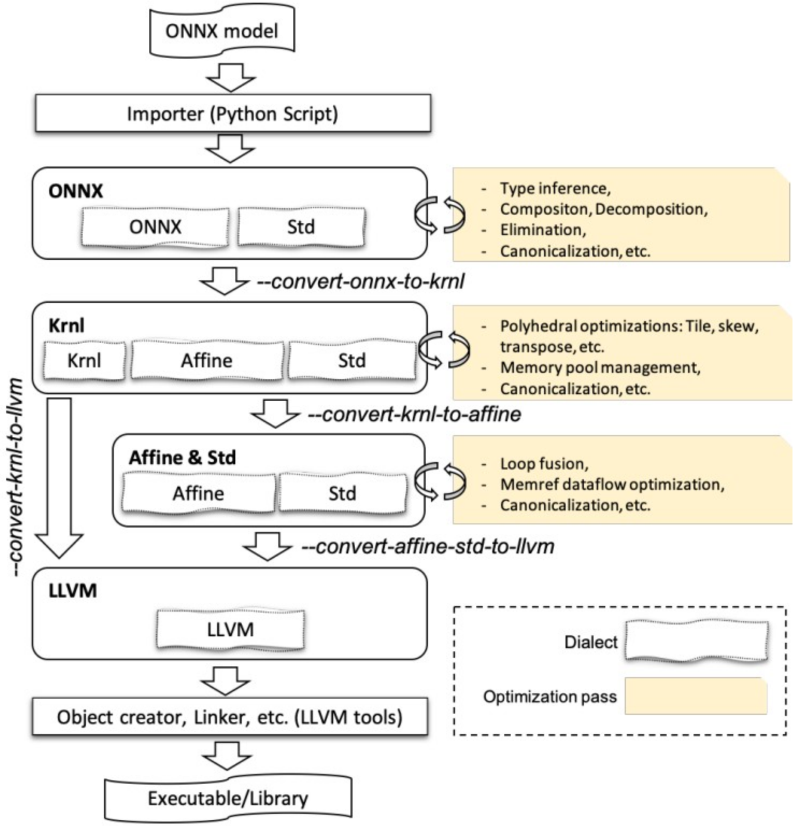


Fig. 2: Architecture of onnx-mlir. Names prefixed with '--' are passes.

Architecture:

- ONNX Dialect:
  - Represent ONNX with mlir
  - Tablegen, Generated by Python script
    - In MLIR, usually you write .td file yourself.
    - You can define the following in .td file:
      - Dialect
      - Operation
      - Attribute
      - Type

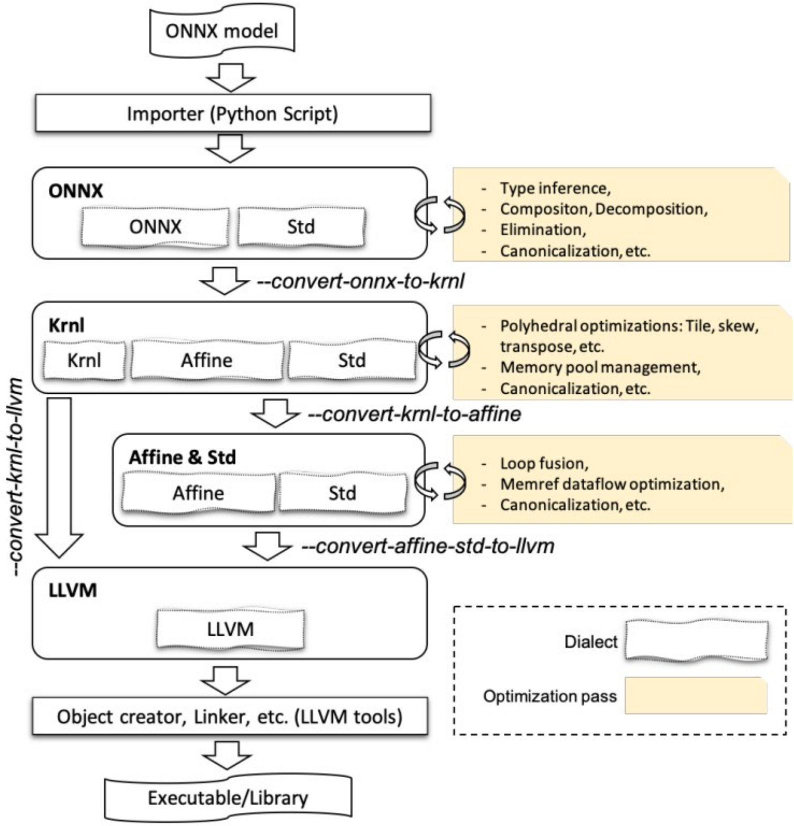


Fig. 2: Architecture of onnx-mlir. Names prefixed with '--' are passes.

Architecture

- ONNX Dialect:
  - Tablegen allow to represent information to human-readable form:
    - Arguments: input & attribute
    - Results: output
  - The ONNX and Tablegen definition are quite similar.

Listing 6: Tablegen-based definition for operation relu.

```
def ONNXLeakyReluOp:ONNX_Op<"LeakyRelu",
  [NoSideEffect, DeclareOpInterfaceMethods<ShapeInferenceOpInterface>]> {
    let summary = "ONNX LeakyRelu operation";
    let description = [{"LeakyRelu takes ... "}];
    let arguments = (ins AnyTypeOf<[TensorOf<[F16]>, TensorOf<[F32]>, TensorOf<[F64]>]>:$X, ↩
        DefaultValuedAttr<F32Attr, "0.01">:$alpha);
    let results = (outs AnyTypeOf<[TensorOf<[F16]>, TensorOf<[F32]>, TenorOf<[F64]>]>:$Y);
    let extraClassDeclaration = [{ ... }];
}
```

Architecture

- ONNX Dialect (LeakyRelu):
  - Real Example: (src/Dialect/ONNX/ONNXOPs.td.inc)
  - Generated by: (utils/gen_onnx_mlir.py)

```
def ONNXLeakyReluOp:ONNX_Op<"LeakyRelu",
  [Pure, DeclareOpInterfaceMethods<ShapeInferenceOpInterface>, DeclareOpInterfaceMethods<ShapeHelperOpInterface>]> {
  let summary = "ONNX LeakyRelu operation";
  let description = [{
  LeakyRelu takes input data (Tensor<T>) and an argument alpha, and produces one
  output data (Tensor<T>) where the function `f(x) = alpha * x for x < 0`,
  `f(x) = x for x >= 0`, is applied to the data tensor elementwise.
  }];
  let arguments = (ins AnyTypeOf<[TensorOf<[BF16]>, TensorOf<[F16]>, TensorOf<[F32]>, TensorOf<[F64]>]>:$X,
    DefaultValuedAttr<F32Attr, "0.01">:$alpha);
  let results = (outs AnyTypeOf<[TensorOf<[BF16]>, TensorOf<[F16]>, TensorOf<[F32]>, TensorOf<[F64]>]>:$Y);
  let extraClassDeclaration = [{
    static int getNumberOfOperands() {
      return 1;
    }
```

- ONNX Dialect (Conv):
  - Real Example: (src/Dialect/ONNX/ONNXOPs.td.inc)

```
def ONNXConvOp:ONNX_Op<"Conv",
  [Pure, DeclareOpInterfaceMethods<ShapeInferenceOpInterface>, DeclareOpInterfaceMethods<ShapeHelperOpInterface>]> {
  let summary = "ONNX Conv operation";
  let description = [{
  The convolution operator consumes an input tensor and a filter, and
  computes the output.
  }];
  let arguments = (ins AnyTypeOf<[TensorOf<[F16]>, TensorOf<[F32]>, TensorOf<[F64]>]>:$X,
    AnyTypeOf<[TensorOf<[F16]>, TensorOf<[F32]>, TensorOf<[F64]>]>:$W,
    AnyTypeOf<[TensorOf<[F16]>, TensorOf<[F32]>, TensorOf<[F64]>, NoneType]>:$B,
    DefaultValuedStrAttr<StrAttr, "NOTSET">:$auto_pad,
    OptionalAttr<I64ArrayAttr>:$dilations,
    DefaultValuedAttr<SI64Attr, "1">:$group,
    OptionalAttr<I64ArrayAttr>:$kernel_shape,
    OptionalAttr<I64ArrayAttr>:$pads,
    OptionalAttr<I64ArrayAttr>:$strides);
  let results = (outs AnyTypeOf<[TensorOf<[F16]>, TensorOf<[F32]>, TensorOf<[F64]>]>:$Y);
  let builders = [
    OpBuilder<(ins "Value":$X, "Value":$W, "Value":$B, "StringAttr":$auto_pad, "ArrayAttr":$dilations, "IntegerAttr":$group,
    "ArrayAttr":$kernel_shape, "ArrayAttr":$pads, "ArrayAttr":$strides), [{
      auto resultType = UnrankedTensorType::get(X.getType().cast<ShapedType>().getElementType());
      build($_builder, $_state, resultType, X, W, B, auto_pad, dilations, group, kernel_shape, pads, strides);
    }]>,
    OpBuilder<(ins "ValueRange":$operands, "ArrayRef<NamedAttribute>":$attributes), [{
      auto resultType = UnrankedTensorType::get(operands[0].getType().cast<ShapedType>().getElementType());
      build($_builder, $_state, {resultType}, operands, attributes);
    }]>
    ];
```

Architecture:

- krnl Dialect:
  - Bridge between onnx & affine dialect.
  - Affine dialect is ready to use.
  - Most computation are loop nest.
    - Represent loops in polyhedral model.
      - Providing optimization chances:
        - Parallelism
        - Cache locality.
        - SIMD, vectorization

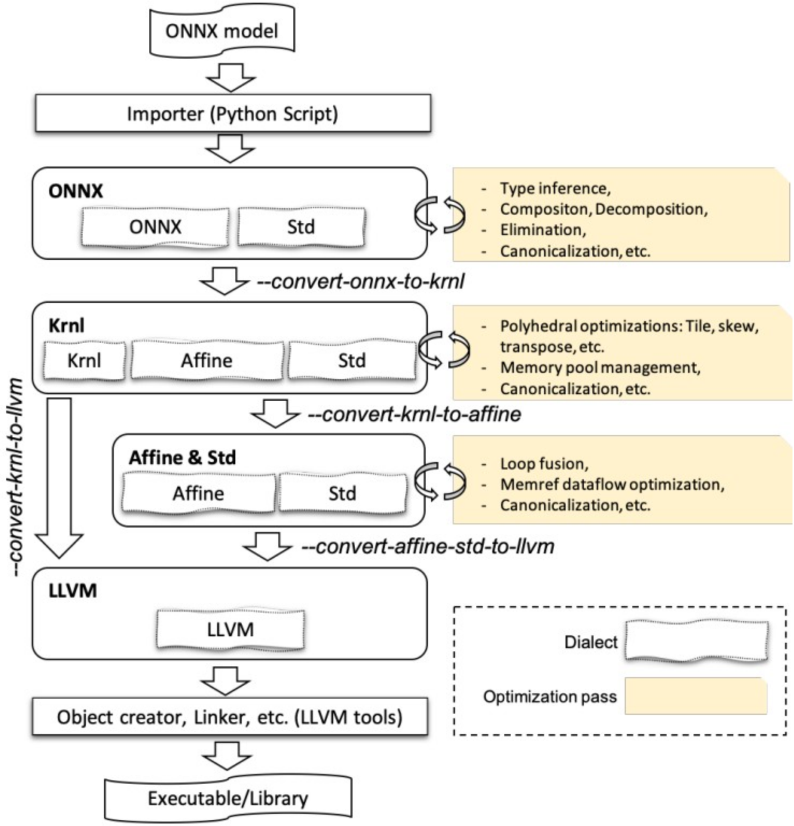


Fig. 2: Architecture of onnx-mlir. Names prefixed with '--' are passes.

Architecture:

- krnl Dialect:
  - At left, Imagine 2 10*10 array added to each other with same array.
  - Now we want to do it with tiling size 2, with a single loop.
  - krnl.iterate (schedule_loop) with (original_loop).
  - Separate program semantics and schedule.

```
%ii, %jj = krnl.define_loops 2
krnl.iterate(%ii, %jj) with (%ii -> %i = 0
    to 10, %jj -> %j = 0 to 10) {
  %foo = std.addi %i, %j : index
}
```

```
%ii = krnl.define_loops 1
%ib, %il = krnl.block %ii 2 : ↩
    (!krnl.loop)->(!krnl.loop, !krnl.loop)
krnl.iterate(%ib, %il) with (%ii -> %i = 0
    to 10) {
  %foo = std.addi %i, %i : index
}
```

Architecture:

- krnl Dialect:
  - --convert-krnl-to-affine pass generates optimized affine.for based loops.
  - The inner loop affine.for is iterating tile size 2.
  - Skew and permutation are applied similarly, and are composable.

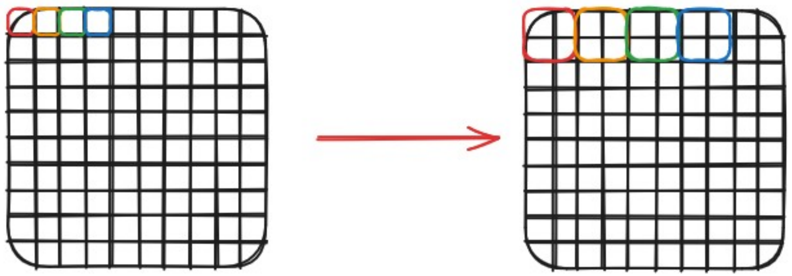
```
%ii = krnl.define_loops 1
%ib, %il = krnl.block %ii 2 : ↩
      (!krnl.loop)->(!krnl.loop, !krnl.loop)
krnl.iterate(%ib, %il) with (%ii -> %i = 0
      to 10) {
  %foo = std.addi %i, %i : index
}
```

```
#map0 = affine_map<(d0) -> (d0)>
#map1 = affine_map<(d0) -> (d0 + 2)>
affine.for %arg0 = 0 to 10 step 2 {
  affine.for %arg1 = #map0(%arg0) to
      #map1(%arg0) {
    %0 = addi %arg1, %arg1 : index
  }
}
```

Architecture:

- But, I still don’t know what just happened?
  - Me, too.
  - This is my speculation.

```
for (int i = 0; i < 10; ++i){
    for (int j = 0; j < 10; ++j)
        result[i][j] = matrix1[i][j] + matrix2[i][j];
}
```



```
for (int i = 0; i < 10; i += 2) {
    for (int j = 0; j < 10; j += 2) {
        for (int ii = i; ii < i + 2; ++ii) {
            for (int jj = j; jj < j + 2; ++jj) {
                result[ii][jj] = matrix1[ii][jj] + matrix2[ii][jj];
            }
        }
    }
}
```

Architecture:

- Affine Dialect:
  - Mainly for:
    - Actual loop transformation.
    - Dependence analysis.
    - Ready-to-use.

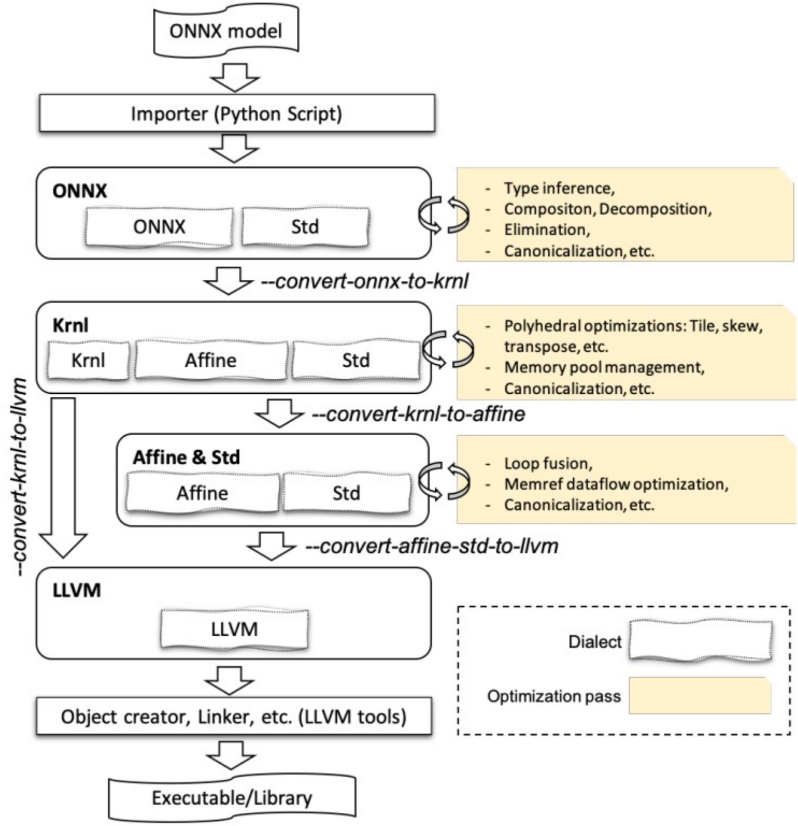


Fig. 2: Architecture of onnx-mlir. Names prefixed with '--' are passes.

Architecture:

- Std Dialect:
  - load, store, addi, addf, absf, call...
  - Obsolete
    - ops are moved to other dialects.

```
%0 = affine.load %arg0[%arg2, %arg3] : memref<3x2xf32>
%1 = affine.load %arg1[%arg2, %arg3] : memref<3x2xf32>
%2 = arith.addf %0, %1 : f32
affine.store %2, %alloc[%arg2, %arg3] : memref<3x2xf32>
```

```
%1 = affine.load %arg0[%arg2, %arg3, %arg4] : memref<3x4x5xf32>
%2 = affine.load %arg1[%arg2, %arg3, %arg4] : memref<3x4x5xf32>
%3 = std.addf %1, %2 : f32
affine.store %3, %0[%arg2, %arg3, %arg4] : memref<3x4x5xf32>
```

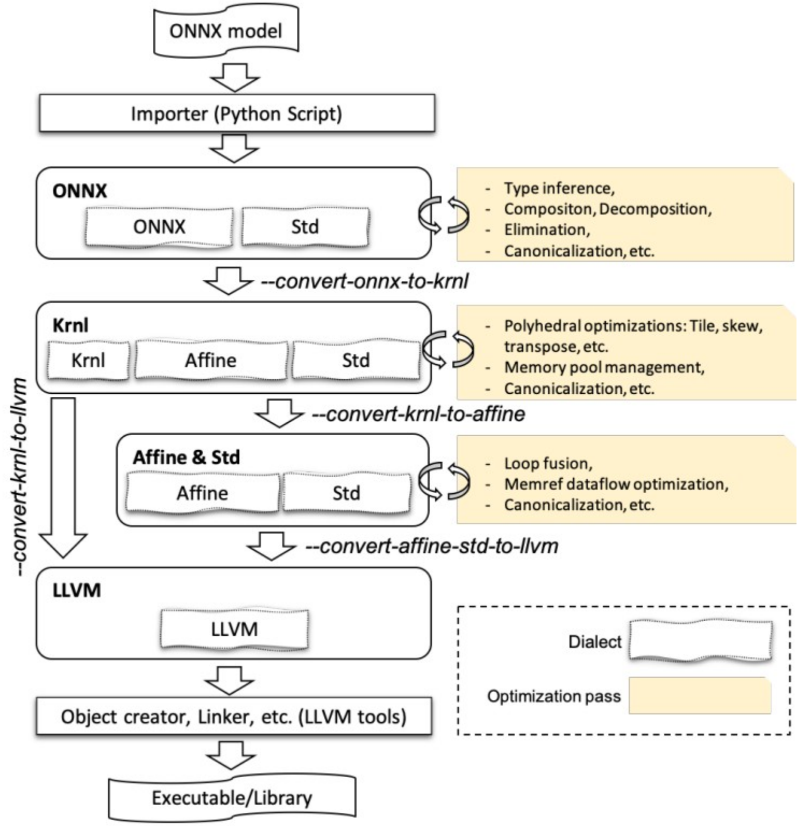


Fig. 2: Architecture of onnx-mlir. Names prefixed with '--' are passes.

Architecture:

- llvm Dialect:
  - Wrapping the LLVM IR into MLIR.
  - Generate bitcode.

```
llvm.func @run_main_graph_llvmir(%arg0: !llvm.ptr) -> !llvm.ptr {
  %0 = llvm.mlir.constant(2 : i64) : i64
  %1 = llvm.mlir.constant(0 : i64) : i64
  %2 = llvm.mlir.constant(1 : i64) : i64
  %3 = llvm.call @omTensorListGetOmtArray(%arg0) : (!llvm.ptr) -> !llvm.ptr
  %4 = llvm.alloca %2 x !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)> : (i64) -> !llvm.ptr
  %5 = llvm.load %3 : !llvm.ptr -> !llvm.ptr
  %6 = llvm.alloca %2 x !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)> : (i64) -> !llvm.ptr
  %7 = llvm.mlir.undef : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %8 = llvm.call @omTensorGetDataPtr(%5) : (!llvm.ptr) -> !llvm.ptr
  %9 = llvm.insertvalue %8, %7[0] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %10 = llvm.insertvalue %8, %9[1] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %11 = llvm.insertvalue %1, %10[2] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %12 = llvm.call @omTensorGetShape(%5) : (!llvm.ptr) -> !llvm.ptr
  %13 = llvm.call @omTensorGetStrides(%5) : (!llvm.ptr) -> !llvm.ptr
  %14 = llvm.load %12 : !llvm.ptr -> i64
  %15 = llvm.insertvalue %14, %11[3, 0] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %16 = llvm.load %13 : !llvm.ptr -> i64
  %17 = llvm.insertvalue %16, %15[4, 0] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %18 = llvm.getelementptr %12[1] : (!llvm.ptr) -> !llvm.ptr, i64
  %19 = llvm.load %18 : !llvm.ptr -> i64
  %20 = llvm.insertvalue %19, %17[3, 1] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %21 = llvm.getelementptr %13[1] : (!llvm.ptr) -> !llvm.ptr, i64
  %22 = llvm.load %21 : !llvm.ptr -> i64
  %23 = llvm.insertvalue %22, %20[4, 1] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  llvm.store %23, %6 : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>, !llvm.ptr
  %24 = llvm.getelementptr %3[1] : (!llvm.ptr) -> !llvm.ptr, !llvm.ptr
  %25 = llvm.load %24 : !llvm.ptr -> !llvm.ptr
  %26 = llvm.alloca %2 x !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)> : (i64) -> !llvm.ptr
  %27 = llvm.mlir.undef : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %28 = llvm.call @omTensorGetDataPtr(%25) : (!llvm.ptr) -> !llvm.ptr
  %29 = llvm.insertvalue %28, %27[0] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %30 = llvm.insertvalue %28, %29[1] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %31 = llvm.insertvalue %1, %30[2] : !llvm.struct<(ptr, ptr, i64, array<2 x i64>, array<2 x i64>)>
  %32 = llvm.call @omTensorGetShape(%25) : (!llvm.ptr) -> !llvm.ptr
```

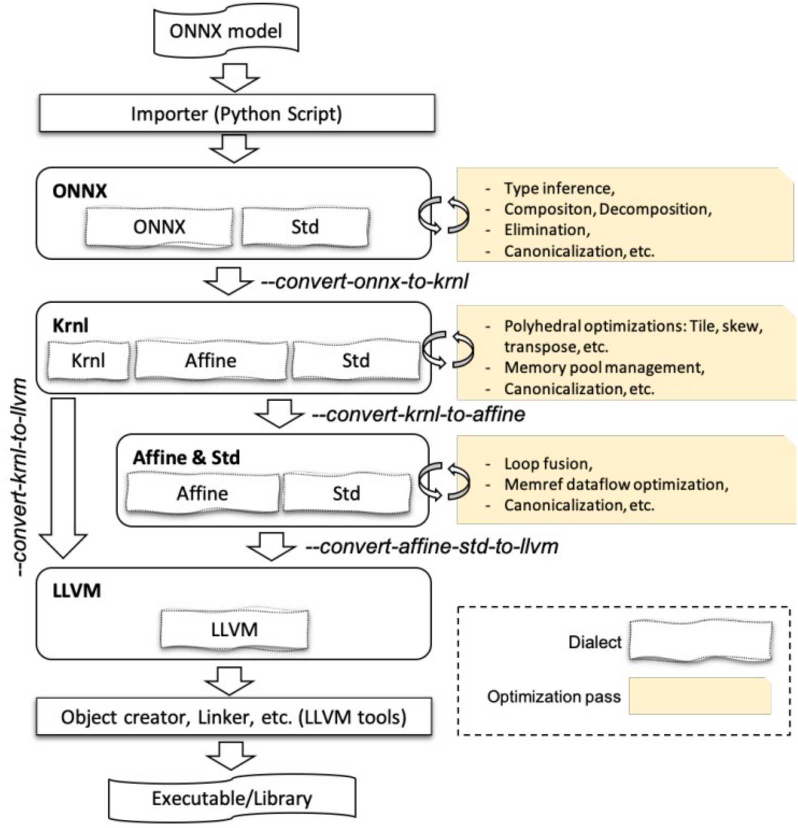


Fig. 2: Architecture of onnx-mlir. Names prefixed with '--' are passes.

Optimization Passes:

- Optimization:
  - Operation decomposition
  - Shape inference
  - Graph rewriting
  - Constant propagation

Optimization Passes:

- Operation decomposition:
  - Breaking down complex operations into a sequence of simpler ones
    - e.g. ReduceL1 operation

```
def ReduceL1Pattern: Pat<
  (ReduceL1Op $x, $axes, $keepdims),
  (ReduceSumOp (AbsOp $x), $axes, $keepdims)
>;
```

src > Transform > ONNX > Decompose.td

```
//===----------------------------------------------------------------------===//
// ONNXReduceL1Op %X = ONNXReduceSumOp (ONNXAbsOp %X)
//===----------------------------------------------------------------------===//
def ReduceL1OpPattern
    : Pat<(ONNXReduceL1Op $oprd, $axes, $keepdims, $noop),
          (ONNXReduceSumOp(ONNXAbsOp $oprd), $axes, $keepdims, $noop)>;

//===----------------------------------------------------------------------===//
// ONNXReduceL2Op %X = ONNXSqrtOp (ONNXReduceSumSquareOp (%X))
//===----------------------------------------------------------------------===//
def ReduceL2OpPattern
    : Pat<(ONNXReduceL2Op $oprd, $axes, $keepdims, $noop),
          (ONNXSqrtOp(ONNXReduceSumSquareOp $oprd, $axes, $keepdims, $noop))>;
```

Optimization Passes:

- Operation decomposition:
  - Breaking down complex operations into a sequence of simpler ones
    - e.g. ReduceL1 operation

L1-norm

$$||x||_1 := \sum_{i=1}^{n} |x_i|$$

- L1-norm 又稱為 Taxicab-norm (計程車幾何範數) 或 Manhattan-norm (曼哈頓距離範數)

L2-norm

$$||x||_2 := [\sum_{i=1}^{n} |x_i|^2]^{1/2} = \sqrt{|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2}$$

Optimization Passes:

- Shape inference:
  - Traverses all operations
  - Infers the shapes of tensors with unrank shapes (e.g. tensor<*xf32>)
  - Propagates the ranked shapes to consuming operations
    - e.g.: A output to B, A is tensor<3*4>, so B is tensor<3*4> too.
  - Terminates until all tensors have ranked shapes.

Listing 3: Operation add in onnx dialect, generated using importer.

```
module {
  func @main_graph( %arg0:tensor<3x4x5xf32>, %arg1:tensor<3x4x5xf32>) -> tensor<*xf32> {
```

Listing 4: Operation add in krnl dialect, generated by applying passes --shape-inference and --convert-onnx-to-krnl.

```
module {
  func @main_graph(%arg0: memref<3x4x5xf32>, %arg1: memref<3x4x5xf32>) -> memref<3x4x5xf32> {
```

Optimization Passes:

- Graph rewriting:
  - Rewriting rules are conveniently represented using DRRs
  - Examples: Fuse add and mul ops together

```
def MulAddToGemmPattern : Pat<
  (AddOp (MatMulOp:$res $m1, $m2), $m3),
  (GemmOp $m1, $m2, $m3),
  [(HasOneUse $res)]
>;
```

src > Dialect > ONNX > Rewrite.td

```
// onnx.add(onnx.matmul(%X, %Y), %Z) = onnx.Gemm(%X, %Y, %Z)
def MulAddToGemmOptPattern : Pat<(ONNXAddOp (ONNXMatMulOp:$res $m1, $m2), $m3),
                                 (ONNXGemmOp $m1, $m2, $m3, (GemmAlpha), (GemmBeta), (GemmTransA), (GemmTransB)),
                                 [(HasOneUse $res), (HasRankOf<2> $m1), (HasRankOf<2> $m2)]>;
```

Optimization Passes:

- Graph rewriting:
  - Rewriting rules are conveniently represented using DRRs
  - Examples: remove identity op

```
def IdentityEliminationPattern : Pat<
  (ONNXIdentityOp $arg),
  (replaceWithValue $arg)
>;
```

src > Dialect > ONNX > Rewrite.td

```
// ONNX_Op (onnx.Identity (%X)) = ONNX_Op (%X)
def IdentityEliminationPattern : Pat<(ONNXIdentityOp $arg),
                                     (replaceWithValue $arg)>;
```

Optimization Passes:

- Constant propagation:
  - Compute the constant together at compile-time.
  - If mixed constant and variable, normalize it:
    - e.g. (onnx.Add)
      - Utillize associativity, group variable and constant respectively.

$$(1)\quad c + x \Rightarrow x + c$$

$$(2)\quad (x + c_1) + c_2 \Rightarrow x + (c_1 + c_2)$$

$$(3)\quad (x + c) + y \Rightarrow (x + y) + c$$

$$(4)\quad x + (y + c) \Rightarrow (x + y) + c$$

$$(5)\quad (x + c_1) + (y + c_2) \Rightarrow (x + y) + (c_1 + c_2)$$

Experiment at the time:

- Support 51/139 ONNX operations.
  - Now: 75/209 not supported.
- Can compile MNIST and Resnet.
  - I can compile efficientnet_v2 (image classification)
- Can run on x86 machine, IBM power systems, system Z.
- Encountered big-endian and little-endian problem.
  - ONNX is little-endian, system Z is big-endian
  - MLIR does not support big-endian well.
  - Patch was able to resolve the problem.

Experiment at the time:

- Benchmark: (2.3-GHz POWER9 processors)
- MNIST: Graph Rewriting is applied.
- ResNet50: No optimization applied.
- Author believe that if polyhedral optimizations, SIMD optimization, and loop fusion is applied, the performance would be improved.

Table 1: Run inferencing with MNIST and ResNet50 on a POWER9 machine. Time in seconds.

Model	Compilation time	Inference time
MNIST	0.237	0.001
ResNet50	7.661	7.540

My Experiment:

- Model: efficientnet_v2
  - 87.9MB
  - Light weight Image Classification model
- Dataset: cifar100
- Compile time: 1m27.788s (single threaded)
- Inference time: 0m1.738s
- What can we get from this information?
  - I don’t know.

My Experiment:

- How did I experiment?
  - Use ImageNet-1k pretrained weight to do transfer learning.
  - Convert the pytorch model to .onnx
  - Compile model with onnx-mlir to generate .so file
  - Rename the PyRuntime.xxx.so to PyRuntime.so
    - Link and export don’t work.
  - Implement the driver.
- The actually used command:

```
torch.onnx.export(model, example_input, onnx_path, verbose=True)
~/onnx-mlir/build/Debug/bin/onnx-mlir --O3 --EmitLib /home/sylvex/dl_cnn_hw1/efficientnet_32.onnx
```

My Experiment:

- How did I implement driver?
  - Python Runtime.
  - docs/mnist_example would help you to write c/python/java driver

```python
import PyRuntime
session = PyRuntime.OMExecutionSession("./efficientnet_32.so")
outputs = session.run([image_array])
prediction = outputs[0]
for i in range(0, 100):
    score = prediction[0, i]
    if score > max_score:
        candidate = i
        max_score = score
    print(f"candidate:{coarse_label[i]} has score:{score}")

print(f"candidate:{coarse_label[candidate]} has max score:{max_score}")
```

My Experiment:

- Result:

Input:  

Output:

```
candidate:tractor has score:-4.397192001342773
candidate:train has score:-4.818047523498535
candidate:trout has score:-2.1939566135406494
candidate:tulip has score:0.8204006552696228
candidate:turtle has score:-5.017601490020752
candidate:wardrobe has score:-1.3327709436416626
candidate:whale has score:-3.0263237953186035
candidate:willow_tree has score:2.586937427520752
candidate:wolf has score:-3.4177582263946533
candidate:woman has score:-2.024754762649536
candidate:worm has score:1.5632660388946533
candidate:orchid has max score:9.20013427734375
```

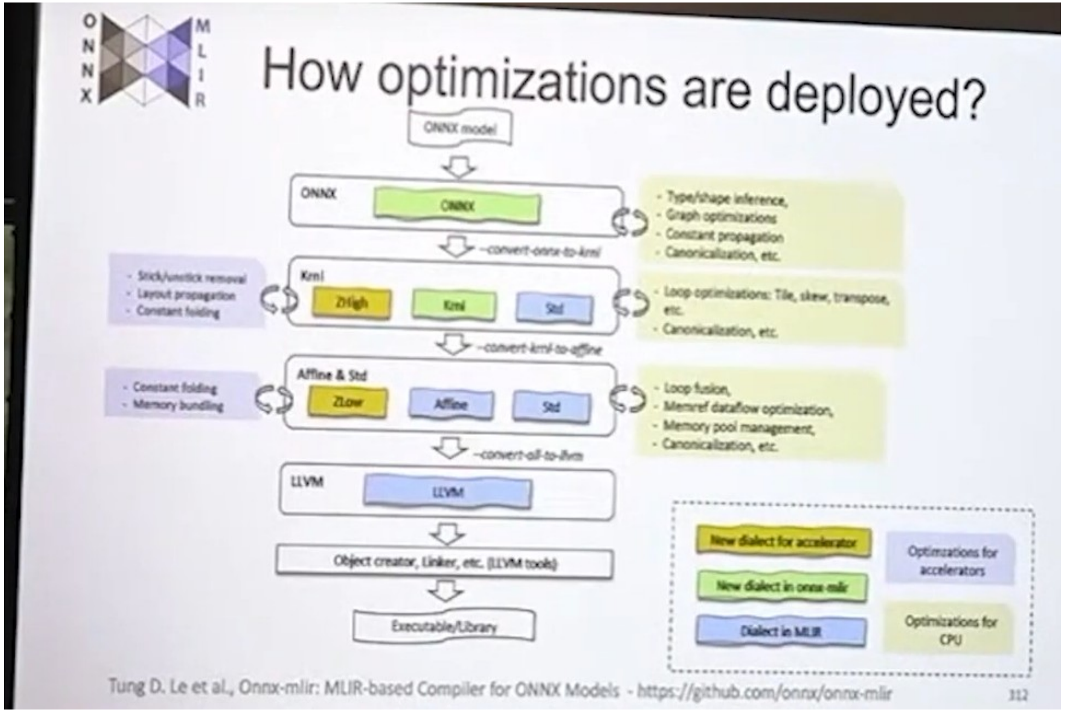
```
candidate:tractor has score:-0.8939839005470276
candidate:train has score:-2.0960440635681152
candidate:trout has score:-2.1811110973358154
candidate:tulip has score:-1.3569916486740112
candidate:turtle has score:-1.5122387409210205
candidate:wardrobe has score:0.15867894887924194
candidate:whale has score:-1.0275558233261108
candidate:willow_tree has score:-2.600288867950439
candidate:wolf has score:6.0673394203186035
candidate:woman has score:7.935925483703613
candidate:worm has score:-0.6879879236221313
candidate:girl has max score:16.172103881835938
```

Paper Conclusion:

- The project transform onnx graph to native code with MLIR.
- Proposed Dialects: onnx, krnl
- Discussed Optimizations Implemented:
  - Operation decomposition, graph rewriting, constant propagation, etc
- Easily integrates new optimizations thanks to MLIR
- Future Optimizations:
  - Polyhedral optimization, Loop fusion, SIMD optimization.
  - Code generation for accelerators.

Research Opportunity:

- How to add accelerator?



Personal thought:

- Prove-of-concept works, project have long way to go.
- Are there optimization chance lost when:
  - Conversion between dialect.
  - Dialect coverage.
  - Weird hardware like 1 operation complete convolution?
- MLIR provide template for building compiler faster, but the core is how you redirect the problem.
- Have to read more about the ecosystem if I research for MLIR.

Reference:

- I forgot.
- I’ve seemed too much stuff, forget to note them respectively.
- Sorry.
- Definitely not lazy.
- Trust me.

Thank you


喂，老師嗎？
我現在在宿舍樓頂
我真的看不完文獻了
這裡風好大
我好害怕


爸爸:期末考考得怎麼樣? 我:
@迷因翻譯
成績出來之前的我: