ONNX

Architecture & Infra SIG Update

Liqun Fu, Microsoft
2023

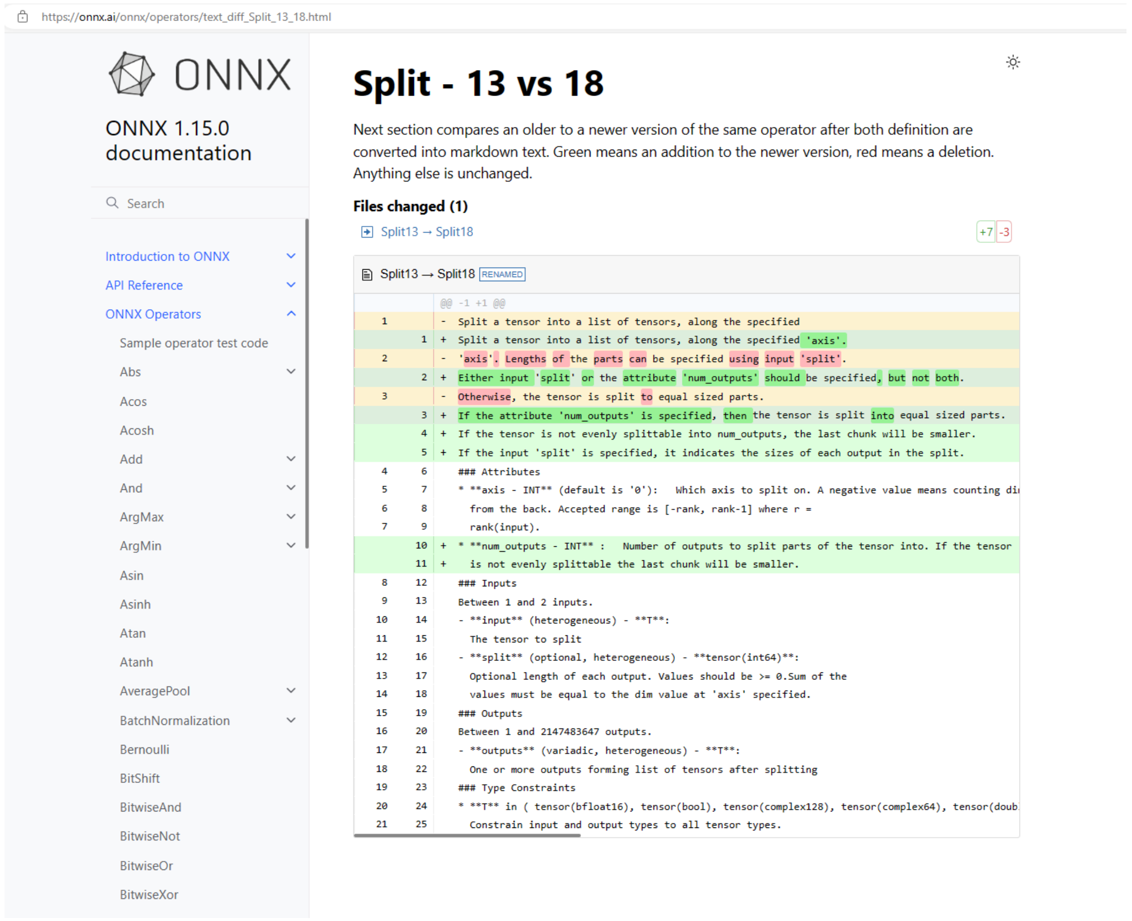
Update

- 2 releases since last meet-up
    - 1.13 (Przemyslaw Wysocki, Intel)
    - 1.14 (Yuan Yao, Nvidia)
- IR 9
    - 8-bit Float
    - More support to local functions
- Reference implementation for ONNX operators

Update (Continued)

- Serialization with Textproto format
- Function Ops can have multiple bodies, each uses a different opset version
- Inline Local functions
- Documentation page: Comparing between versions of an operator

Highlight Op changes on the New Documentation page



https://onnx.ai/onnx/operators/text_diff_Split_13_18.html

Split - 13 vs 18

Next section compares an older to a newer version of the same operator after both definition are converted into markdown text. Green means an addition to the newer version, red means a deletion. Anything else is unchanged.

Files changed (1)

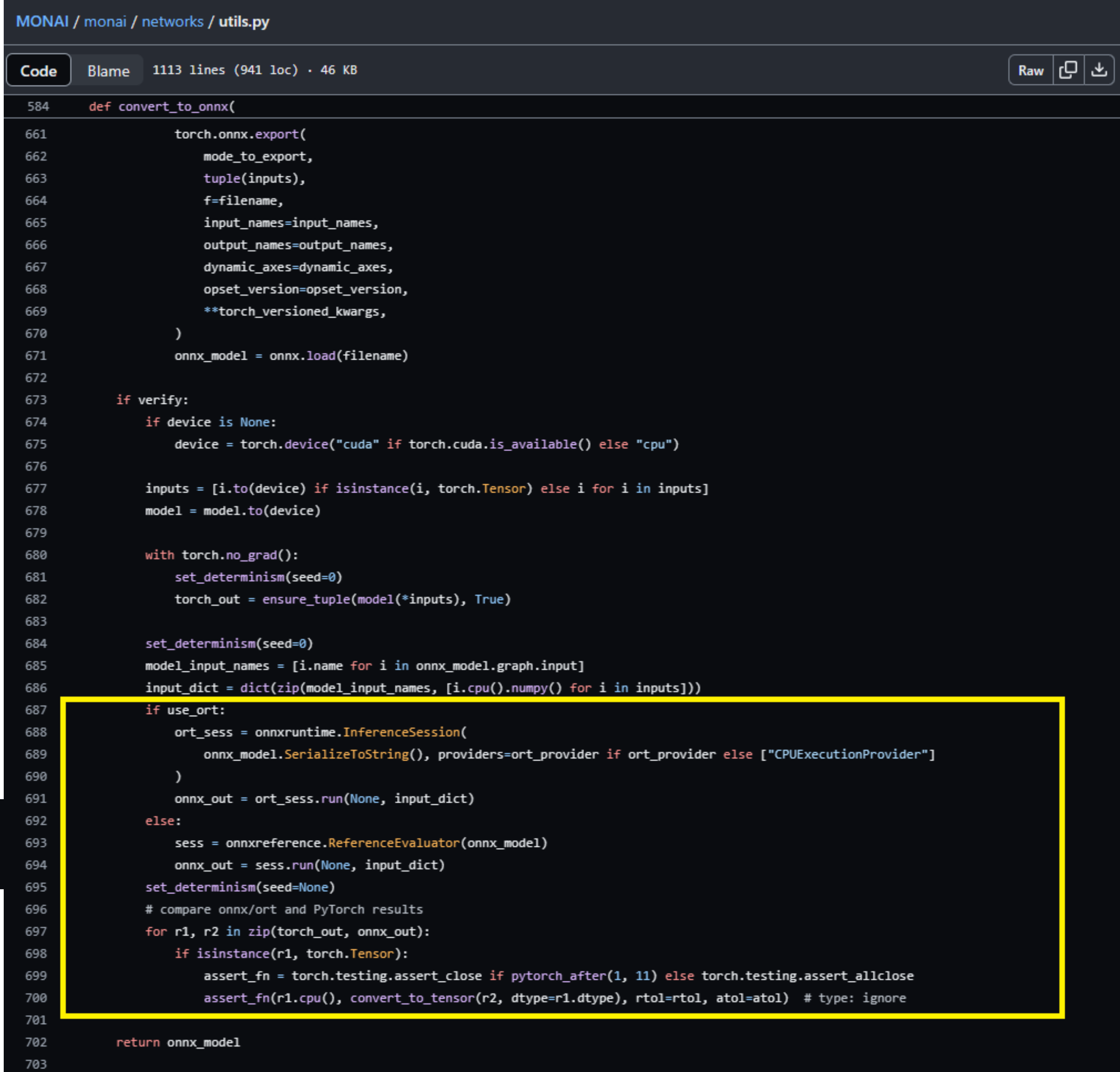
Split13 → Split18 +7 -3

Split13 → Split18 RENAMED

```
@@ -1 +1 @@
- Split a tensor into a list of tensors, along the specified
+ Split a tensor into a list of tensors, along the specified 'axis'.
- 'axis'. Lengths of the parts can be specified using input 'split'.
+ Either input 'split' or the attribute 'num_outputs' should be specified, but not both.
- Otherwise, the tensor is split to equal sized parts.
+ If the attribute 'num_outputs' is specified, then the tensor is split into equal sized parts.
+ If the tensor is not evenly splittable into num_outputs, the last chunk will be smaller.
+ If the input 'split' is specified, it indicates the sizes of each output in the split.
  ### Attributes
  * **axis - INT** (default is '0'):   Which axis to split on. A negative value means counting di
    from the back. Accepted range is [-rank, rank-1] where r =
    rank(input).
+ * **num_outputs - INT** :   Number of outputs to split parts of the tensor into. If the tensor
+   is not evenly splittable the last chunk will be smaller.
  ### Inputs
  Between 1 and 2 inputs.
  - **input** (heterogeneous) - **T**:
    The tensor to split
  - **split** (optional, heterogeneous) - **tensor(int64)**:
    Optional length of each output. Values should be >= 0.Sum of the
    values must be equal to the dim value at 'axis' specified.
  ### Outputs
  Between 1 and 2147483647 outputs.
  - **outputs** (variadic, heterogeneous) - **T**:
    One or more outputs forming list of tensors after splitting
  ### Type Constraints
  * **T** in ( tensor(bfloat16), tensor(bool), tensor(complex128), tensor(complex64), tensor(doub
    Constrain input and output types to all tensor types.
```

Validate PyTorch Converter with ONNX Reference Implementation

```python
sess = onnxreference.ReferenceEvaluator(onnx_model)
onnx_out = sess.run(None, input_dict)
```

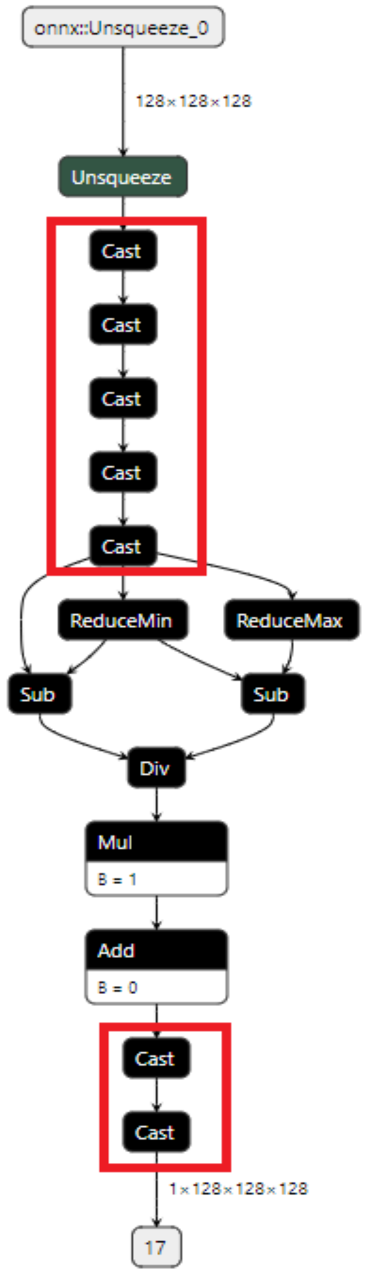


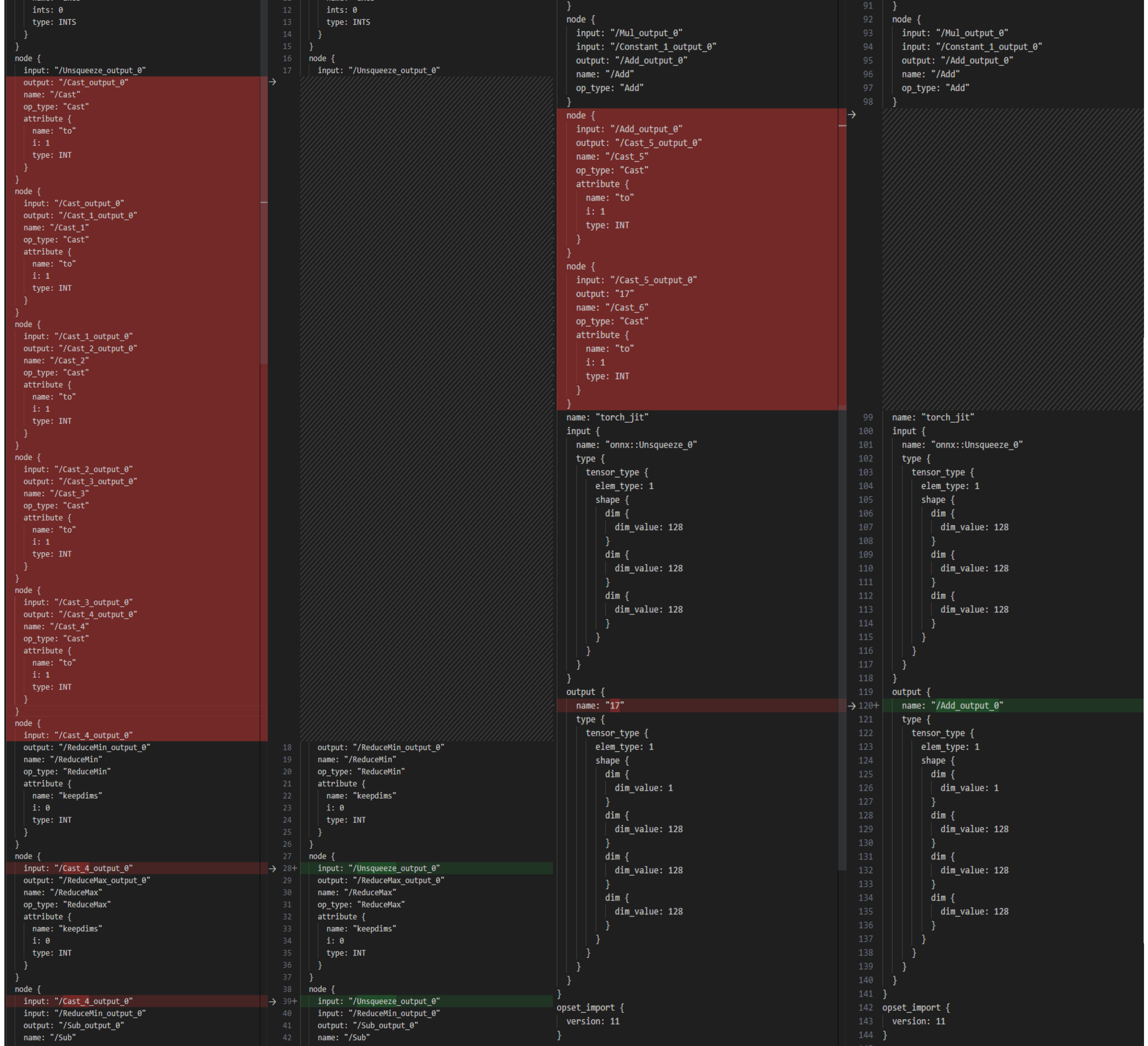
MONAI / monai / networks / utils.py

Code Blame 1113 lines (941 loc) · 46 KB Raw

```python
def convert_to_onnx(
            torch.onnx.export(
                mode_to_export,
                tuple(inputs),
                f=filename,
                input_names=input_names,
                output_names=output_names,
                dynamic_axes=dynamic_axes,
                opset_version=opset_version,
                **torch_versioned_kwargs,
            )
            onnx_model = onnx.load(filename)

    if verify:
        if device is None:
            device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        inputs = [i.to(device) if isinstance(i, torch.Tensor) else i for i in inputs]
        model = model.to(device)

        with torch.no_grad():
            set_determinism(seed=0)
            torch_out = ensure_tuple(model(*inputs), True)

        set_determinism(seed=0)
        model_input_names = [i.name for i in onnx_model.graph.input]
        input_dict = dict(zip(model_input_names, [i.cpu().numpy() for i in inputs]))
        if use_ort:
            ort_sess = onnxruntime.InferenceSession(
                onnx_model.SerializeToString(), providers=ort_provider if ort_provider else ["CPUExecutionProvider"]
            )
            onnx_out = ort_sess.run(None, input_dict)
        else:
            sess = onnxreference.ReferenceEvaluator(onnx_model)
            onnx_out = sess.run(None, input_dict)
        set_determinism(seed=None)
        # compare onnx/ort and PyTorch results
        for r1, r2 in zip(torch_out, onnx_out):
            if isinstance(r1, torch.Tensor):
                assert_fn = torch.testing.assert_close if pytorch_after(1, 11) else torch.testing.assert_allclose
                assert_fn(r1.cpu(), convert_to_tensor(r2, dtype=r1.dtype), rtol=rtol, atol=atol)  # type: ignore

    return onnx_model
```

Use Text Editor To Edit An ONNX Model



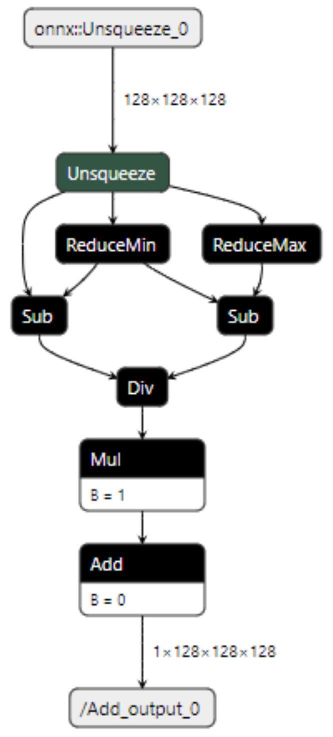


```
      ints: 0
      type: INTS
    }
  }
  node {
    input: "/Unsqueeze_output_0"
    output: "/Cast_output_0"
    name: "/Cast"
    op_type: "Cast"
    attribute {
      name: "to"
      i: 1
      type: INT
    }
  }
  node {
    input: "/Cast_output_0"
    output: "/Cast_1_output_0"
    name: "/Cast_1"
    op_type: "Cast"
    attribute {
      name: "to"
      i: 1
      type: INT
    }
  }
  node {
    input: "/Cast_1_output_0"
    output: "/Cast_2_output_0"
    name: "/Cast_2"
    op_type: "Cast"
    attribute {
      name: "to"
      i: 1
      type: INT
    }
  }
  node {
    input: "/Cast_2_output_0"
    output: "/Cast_3_output_0"
    name: "/Cast_3"
    op_type: "Cast"
    attribute {
      name: "to"
      i: 1
      type: INT
    }
  }
  node {
    input: "/Cast_3_output_0"
    output: "/Cast_4_output_0"
    name: "/Cast_4"
    op_type: "Cast"
    attribute {
      name: "to"
      i: 1
      type: INT
    }
  }
  node {
    input: "/Cast_4_output_0"
    output: "/ReduceMin_output_0"
    name: "/ReduceMin"
    op_type: "ReduceMin"
    attribute {
      name: "keepdims"
      i: 0
      type: INT
    }
  }
  node {
    input: "/Cast_4_output_0"
    output: "/ReduceMax_output_0"
    name: "/ReduceMax"
    op_type: "ReduceMax"
    attribute {
      name: "keepdims"
      i: 0
      type: INT
    }
  }
  node {
    input: "/Cast_4_output_0"
    input: "/ReduceMin_output_0"
    output: "/Sub_output_0"
    name: "/Sub"
```

```
      ints: 0
      type: INTS
    }
  }
  node {
    input: "/Unsqueeze_output_0"
    output: "/ReduceMin_output_0"
    name: "/ReduceMin"
    op_type: "ReduceMin"
    attribute {
      name: "keepdims"
      i: 0
      type: INT
    }
  }
  node {
    input: "/Unsqueeze_output_0"
    output: "/ReduceMax_output_0"
    name: "/ReduceMax"
    op_type: "ReduceMax"
    attribute {
      name: "keepdims"
      i: 0
      type: INT
    }
  }
  node {
    input: "/Unsqueeze_output_0"
    input: "/ReduceMin_output_0"
    output: "/Sub_output_0"
    name: "/Sub"
```

```
  }
  node {
    input: "/Mul_output_0"
    input: "/Constant_1_output_0"
    output: "/Add_output_0"
    name: "/Add"
    op_type: "Add"
  }
  node {
    input: "/Add_output_0"
    output: "/Cast_5_output_0"
    name: "/Cast_5"
    op_type: "Cast"
    attribute {
      name: "to"
      i: 1
      type: INT
    }
  }
  node {
    input: "/Cast_5_output_0"
    output: "17"
    name: "/Cast_6"
    op_type: "Cast"
    attribute {
      name: "to"
      i: 1
      type: INT
    }
  }
  name: "torch_jit"
  input {
    name: "onnx::Unsqueeze_0"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
            dim_value: 128
          }
          dim {
            dim_value: 128
          }
          dim {
            dim_value: 128
          }
        }
      }
    }
  }
  output {
    name: "17"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
            dim_value: 1
          }
          dim {
            dim_value: 128
          }
          dim {
            dim_value: 128
          }
          dim {
            dim_value: 128
          }
        }
      }
    }
  }
}
opset_import {
  version: 11
}
```

```
  }
  node {
    input: "/Mul_output_0"
    input: "/Constant_1_output_0"
    output: "/Add_output_0"
    name: "/Add"
    op_type: "Add"
  }
  name: "torch_jit"
  input {
    name: "onnx::Unsqueeze_0"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
            dim_value: 128
          }
          dim {
            dim_value: 128
          }
          dim {
            dim_value: 128
          }
        }
      }
    }
  }
  output {
    name: "/Add_output_0"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
            dim_value: 1
          }
          dim {
            dim_value: 128
          }
          dim {
            dim_value: 128
          }
          dim {
            dim_value: 128
          }
        }
      }
    }
  }
}
opset_import {
  version: 11
}
```



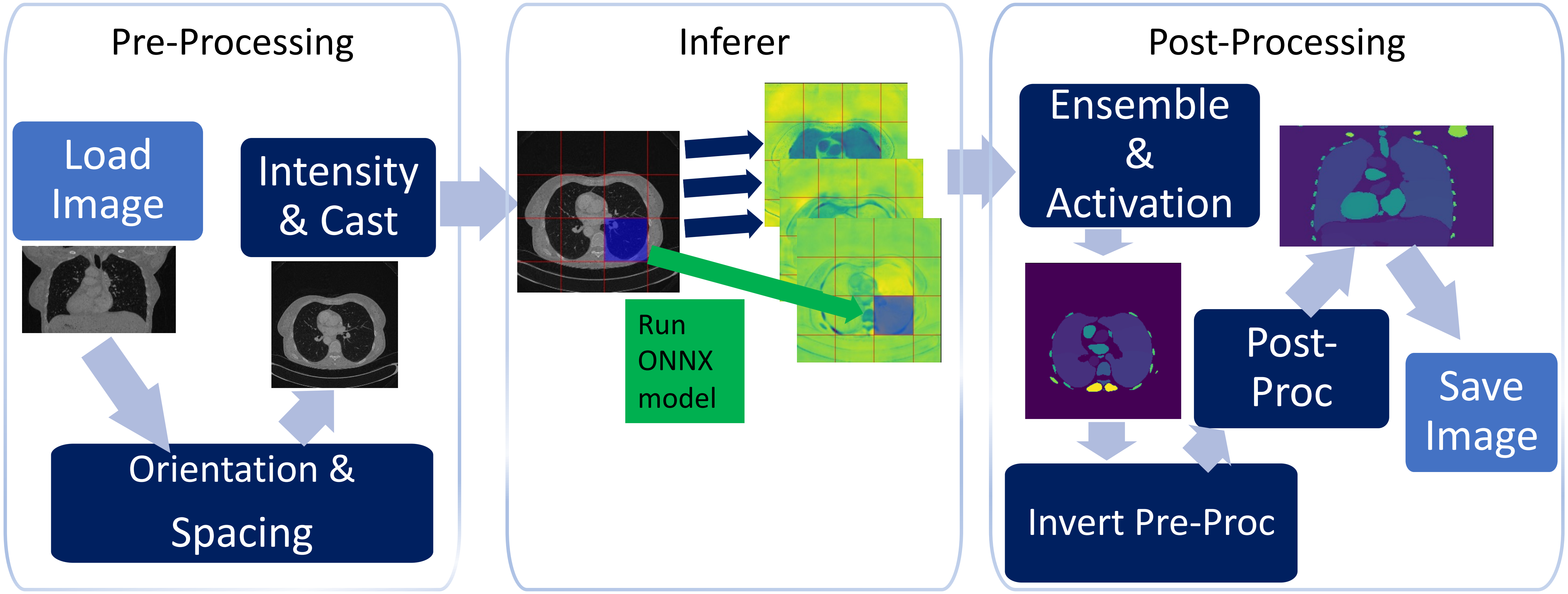
Roadmap

- Maintain a stable CI Infrastructure
- Translate more operators to function op
- Work with operator, pre/post processing, and converter SIGs to support real world models, in production

ONNX with Project MONAI

- https://github.com/Project-MONAI
- Home to most PyTorch models in medical imaging domain
- 20+ SOTA models published in Model-zoo
- ONNX utility to convert all models to ONNX format
- Utilize ORT and ONNX reference implementation for model validation
- However, running core ONNX models is only part of medical imaging workflow
- Need to put end-to-end MONAI workflow in ONNX so it can be applied anywhere on any hardware

Medical Imaging Workflow



ONNX Pre and Post Processing

- Pre- and Post-Processing operations in MONAI are performed using the "Compose" class, which encapsulates transform sequences.
- These operations are wrapped into a Torch module and exported.
- The 'export_compose' function handles exporting the Compose object to ONNX

```
def export_compose(pnp_compose: Compose, opset_version: int, inputs: Sequence[np.ndarray], outputs: Sequence[np.ndarray], image_meta_dict: Dict[str, Any], task_name: str) -> ModelProto:
```

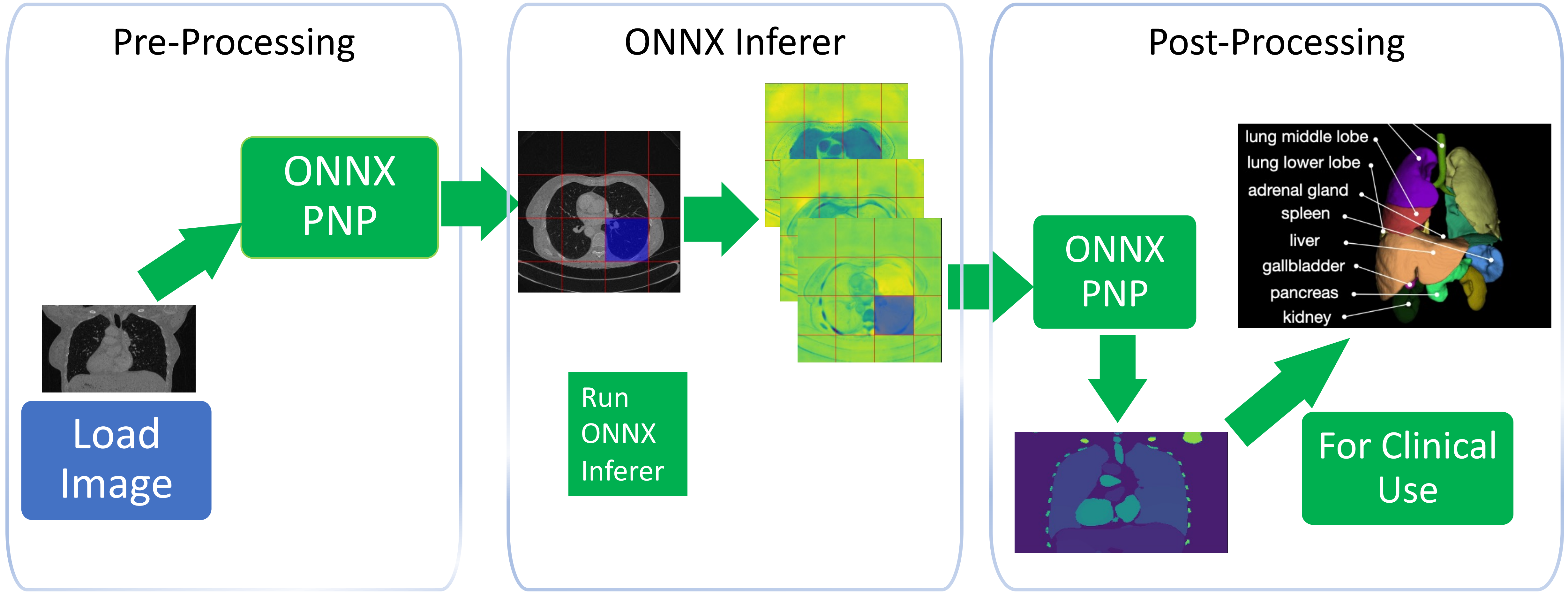
- If new Ops need to be added, we will make it function op. (For example: AffineGrid.)

ONNX Inferer

- Work with SlidingWindowInferer first – it is used by majority of models in model-zoo
- With ONNX-Script, the op can be easily implemented as a function op.
- Combining with ONNX PNP, we complete medical imaging workflow in ONNX.
- Any models developed with MONAI framework can be converted to ONNX automatically

```
@script()
def sliding_window_inference(inputs: FLOAT["N", "C", "D", "H", "W"], roi_size: INT64[3]) -> FLOAT["N", "Seg_C", "D", "H",
"W"]:
    """
    The sliding window method is used for model inference. It involves taking a 3D sliding window on the input tensor
    and making predictions using a provided predictor. The outputs from the predictor are then aggregated to form the
output of the operator.
    """
    inputs_shape = op.Shape(inputs)
    inputs_spatial_shape = op.Shape(inputs, from=2)
    N, _, D, H, W = op.Split(inputs_shape, num_outputs=5)
    roi_D, roi_H, roi_W = op.Split(roi_size, num_outputs=3)

    scan_interval = roi_size
    slices = dense_patch_slices_script(inputs_spatial_shape, roi_size, scan_interval)
    S_, _, _ = op.Split(op.Shape(slices), num_outputs=3)
    S = op.Squeeze(S_, op.Constant(value_ints=[0]))

    seg_C = op.Constant(value_ints=[2]) # TODO: get from predictor model
    output_shape = op.Concat(N, seg_C, inputs_spatial_shape, axis=0)

    aggrregated_pred = op.CastLike(op.ConstantOfShape(output_shape), inputs)
    aggrregated_count = op.CastLike(op.ConstantOfShape(inputs_shape), roi_size)
    for slice_g in range(S):
        win_data, start, stop = prepare_for_predictor_batch_size_is_1_script(inputs, slice_g, slices)
        pred = op.OpaqueOp(win_data, model_path="C:/Temp/sliding_window_predictor_sw_batch_size_is_1.onnx")
        aggrregated_pred, aggrregated_count = aggrregate_predictor_output(pred, start, stop, aggrregated_pred,
aggrregated_count)

    return aggrregated_pred / op.CastLike(aggrregated_count, aggrregated_pred)
```

Complete Medical Imaging Workflow with ONNX



Please Get Involved!

Github: PRs, Issues, and Discussions

Slack channel: https://slack.lfai.foundation and join onnx-archinfra

Monthly SIG meetings (see slack channel for announcements)

Thanks you!

Useful links

- Comparing ONNX operator Split - 13 vs 18
- Load and Save ONNX models in textproto format
- ONNX exporter in MONAI. It uses ONNX reference implementation to validate the converted models
- Affine, GridSample PRs for image pre- and post-processing
- SlidingWindowInferer