



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

 ONNX

ONNX 1.15.0 documentation

Search

Introduction to ONNX

API Reference

ONNX Operators

Sample operator test code

Abs

Acos

Acosh

Add

And

ArgMax

ArgMin

Asin

Asinh

Atan

Atanh

AveragePool

BatchNormalization

Bernoulli

BitShift

BitwiseAnd

BitwiseNot

BitwiseOr

BitwiseXor

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Next section compares an older to a newer version of the same operator after both definition are converted into markdown text.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 [REMOVED]
@@ -1 +1 @@
1 - Split a tensor into a list of tensors, along the specified
1 + Split a tensor into a list of tensors, along the specified 'axis'.
2 - 'axis'. Lengths of the parts can be specified using input 'split'.
2 + Either input 'split' or the attribute 'num_outputs' should be specified, but not both.
3 - Otherwise, the tensor is split to equal sized parts.
3 + If the attribute 'num_outputs' is specified, then the tensor is split into equal sized parts.
4 + If the tensor is not evenly splittable into num_outputs, the last chunk will be smaller.
5 + If the input 'split' is specified, it indicates the sizes of each output in the split.
4 6   ## Attributes
5 7   * *axis - INT* (default is '0'): Which axis to split on. A negative value means counting di
6 8   from the back. Accepted range is [-rank, rank-1] where r =
7 9   rank(input).
10 + * *num_outputs - INT* : Number of outputs to split parts of the tensor into. If the tensor
11 + is not evenly splittable the last chunk will be smaller.
8 12  ## Inputs
9 13  Between 1 and 2 inputs.
10 14 - **input** (heterogeneous) - **T** :
11 15  The tensor to split
12 16 - **split** (optional, heterogeneous) - **tensor(int64)** :
13 17  Optional length of each output. Values should be >= 0. Sum of the
14 18  values must be equal to the dim value at 'axis' specified.
15 19  ## Outputs
16 20  Between 1 and 2147483647 outputs.
17 21 - **outputs** (variadic, heterogeneous) - **T** :
18 22  One or more outputs forming list of tensors after splitting
19 23  ## Type Constraints
20 24 * **T** in ( tensor(bfloat16), tensor(bool), tensor(complex128), tensor(complex64), tensor(doub
21 25  Constrain input and output types to all tensor types.
```

Validate PyTorch Converter with ONNX Reference Implementation

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

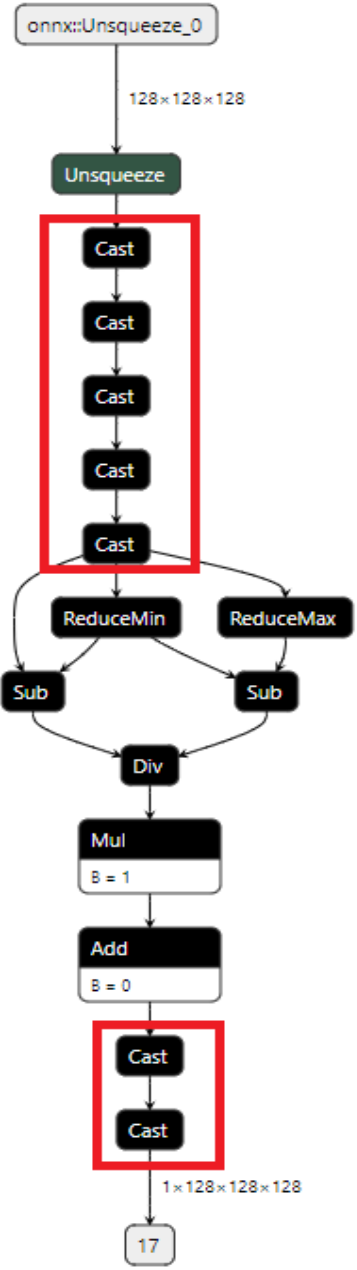
MONAI / monai / networks / utils.py

Code

Blame 1113 lines (941 loc) · 46 KB

Raw Copy Download

```
584 def convert_to_onnx(
661     torch.onnx.export(
662         mode_to_export,
663         tuple(inputs),
664         f=filename,
665         input_names=input_names,
666         output_names=output_names,
667         dynamic_axes=dynamic_axes,
668         opset_version=opset_version,
669         **torch_versioned_kwargs,
670     )
671     onnx_model = onnx.load(filename)
672
673     if verify:
674         if device is None:
675             device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
676
677         inputs = [i.to(device) if isinstance(i, torch.Tensor) else i for i in inputs]
678         model = model.to(device)
679
680         with torch.no_grad():
681             set_determinism(seed=0)
682             torch_out = ensure_tuple(model(*inputs), True)
683
684         set_determinism(seed=0)
685         model_input_names = [i.name for i in onnx_model.graph.input]
686         input_dict = dict(zip(model_input_names, [i.cpu().numpy() for i in inputs]))
687     if use_ort:
688         ort_sess = onnxruntime.InferenceSession(
689             onnx_model.SerializeToString(), providers=ort_provider if ort_provider else ["CPUExecutionProvider"]
690         )
691         onnx_out = ort_sess.run(None, input_dict)
692     else:
693         sess = onnxreference.ReferenceEvaluator(onnx_model)
694         onnx_out = sess.run(None, input_dict)
695     set_determinism(seed=None)
696     # compare onnx/ort and PyTorch results
697     for r1, r2 in zip(torch_out, onnx_out):
698         if isinstance(r1, torch.Tensor):
699             assert_fn = torch.testing.assert_close if pytorch_after(1, 11) else torch.testing.assert_allclose
700             assert_fn(r1.cpu(), convert_to_tensor(r2, dtype=r1.dtype), rtol=rtol, atol=atol) # type: ignore
701
702     return onnx_model
703
```



```

11 int: 0
12 type: INTS
13 }
14 node {
15   input: "/Unsqueeze_output_0"
16   output: "/Cast_output_0"
17   name: "/Cast"
18   op_type: "Cast"
19   attribute {
20     name: "to"
21     i: 1
22     type: INT
23   }
24 }
25 node {
26   input: "/Cast_output_0"
27   output: "/Cast_1_output_0"
28   name: "/Cast_1"
29   op_type: "Cast"
30   attribute {
31     name: "to"
32     i: 1
33     type: INT
34   }
35 }
36 node {
37   input: "/Cast_1_output_0"
38   output: "/Cast_2_output_0"
39   name: "/Cast_2"
40   op_type: "Cast"
41   attribute {
42     name: "to"
43     i: 1
44     type: INT
45   }
46 }
47 node {
48   input: "/Cast_2_output_0"
49   output: "/Cast_3_output_0"
50   name: "/Cast_3"
51   op_type: "Cast"
52   attribute {
53     name: "to"
54     i: 1
55     type: INT
56   }
57 }
58 node {
59   input: "/Cast_3_output_0"
60   output: "/Cast_4_output_0"
61   name: "/Cast_4"
62   op_type: "Cast"
63   attribute {
64     name: "to"
65     i: 1
66     type: INT
67   }
68 }
69 node {
70   input: "/Cast_4_output_0"
71   output: "/ReduceMin_output_0"
72   name: "/ReduceMin"
73   op_type: "ReduceMin"
74   attribute {
75     name: "keepdims"
76     i: 0
77     type: INT
78   }
79 }
80 node {
81   input: "/Cast_4_output_0"
82   output: "/ReduceMax_output_0"
83   name: "/ReduceMax"
84   op_type: "ReduceMax"
85   attribute {
86     name: "keepdims"
87     i: 0
88     type: INT
89   }
90 }
91 node {
92   input: "/ReduceMin_output_0"
93   output: "/Sub_output_0"
94   name: "/Sub"
95   op_type: "Sub"
96   attribute {
97     name: "to"
98     i: 1
99     type: INT
100   }
101 }
102 node {
103   input: "/ReduceMax_output_0"
104   output: "/Sub_output_0"
105   name: "/Sub"
106   op_type: "Sub"
107   attribute {
108     name: "to"
109     i: 1
110     type: INT
111   }
112 }
113 node {
114   input: "/Sub_output_0"
115   output: "/Div_output_0"
116   name: "/Div"
117   op_type: "Div"
118   attribute {
119     name: "to"
120     i: 1
121     type: INT
122   }
123 }
124 node {
125   input: "/Div_output_0"
126   output: "/Mul_output_0"
127   name: "/Mul"
128   op_type: "Mul"
129   attribute {
130     name: "B"
131     value: 1
132     type: INT
133   }
134 }
135 node {
136   input: "/Mul_output_0"
137   output: "/Add_output_0"
138   name: "/Add"
139   op_type: "Add"
140   attribute {
141     name: "B"
142     value: 0
143     type: INT
144   }
145 }
146 node {
147   input: "/Add_output_0"
148   output: "/Cast_output_0"
149   name: "/Cast"
150   op_type: "Cast"
151   attribute {
152     name: "to"
153     i: 1
154     type: INT
155   }
156 }
157 node {
158   input: "/Cast_output_0"
159   output: "/Cast_1_output_0"
160   name: "/Cast_1"
161   op_type: "Cast"
162   attribute {
163     name: "to"
164     i: 1
165     type: INT
166   }
167 }
168 node {
169   input: "/Cast_1_output_0"
170   output: "/Unsqueeze_output_0"
171   name: "/Unsqueeze"
172   op_type: "Unsqueeze"
173   attribute {
174     name: "to"
175     i: 1
176     type: INT
177   }
178 }
179 }

```

```

18 int: 0
19 type: INTS
20 }
21 node {
22   input: "/Unsqueeze_output_0"
23   output: "/Cast_output_0"
24   name: "/Cast"
25   op_type: "Cast"
26   attribute {
27     name: "to"
28     i: 1
29     type: INT
30   }
31 }
32 node {
33   input: "/Cast_output_0"
34   output: "/Cast_1_output_0"
35   name: "/Cast_1"
36   op_type: "Cast"
37   attribute {
38     name: "to"
39     i: 1
40     type: INT
41   }
42 }
33 }
34 }
35 }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }

```

```

91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }

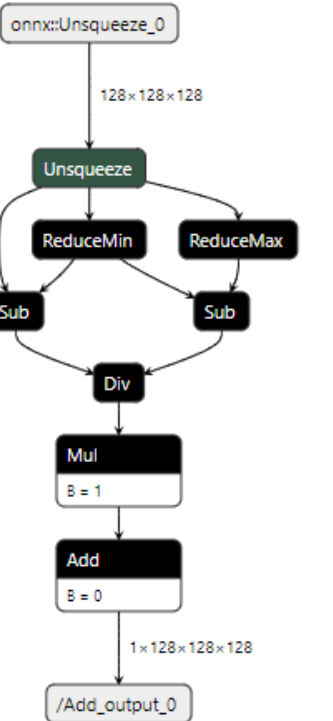
```

```

91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }

```

Use Text Editor To Edit An ONNX Model



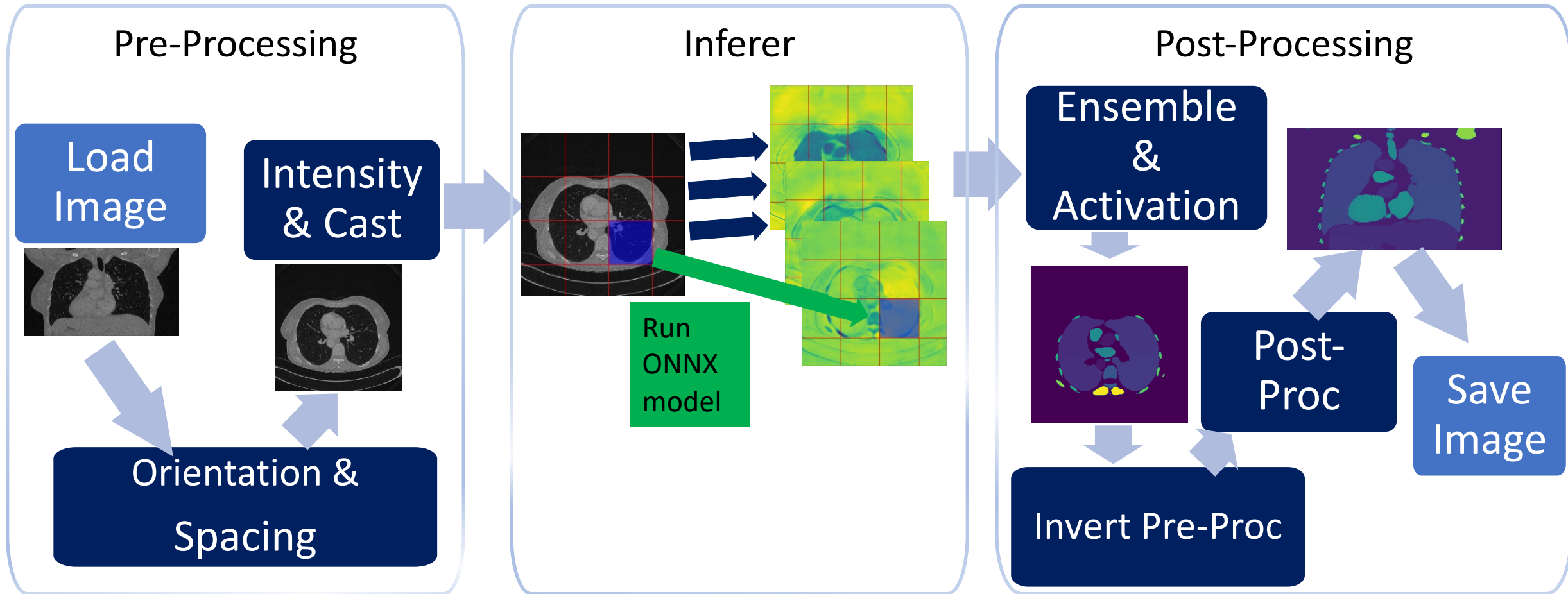
Roadmap

- 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]))

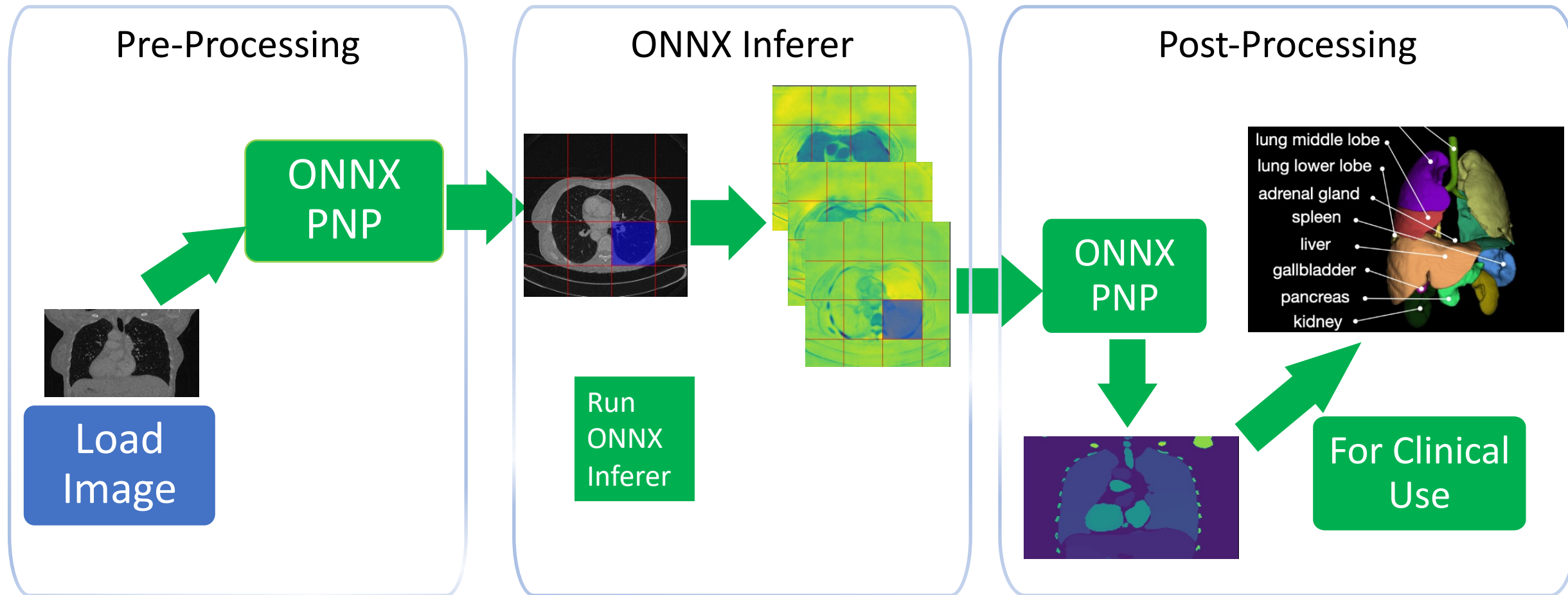
    seg_C = op.Constant(value_ints=[2]) # TODO: get from predictor model
    output_shape = op.Concat(N, seg_C, inputs_spatial_shape, axis=0)

    aggrragated_pred = op.CastLike(op.ConstantOfShape(output_shape), inputs)
    aggrragated_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")
        aggrragated_pred, aggrragated_count = aggregate_predictor_output(pred, start, stop, aggrragated_pred,
aggrragated_count)

    return aggrragated_pred / op.CastLike(aggrragated_count, aggrragated_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](#)