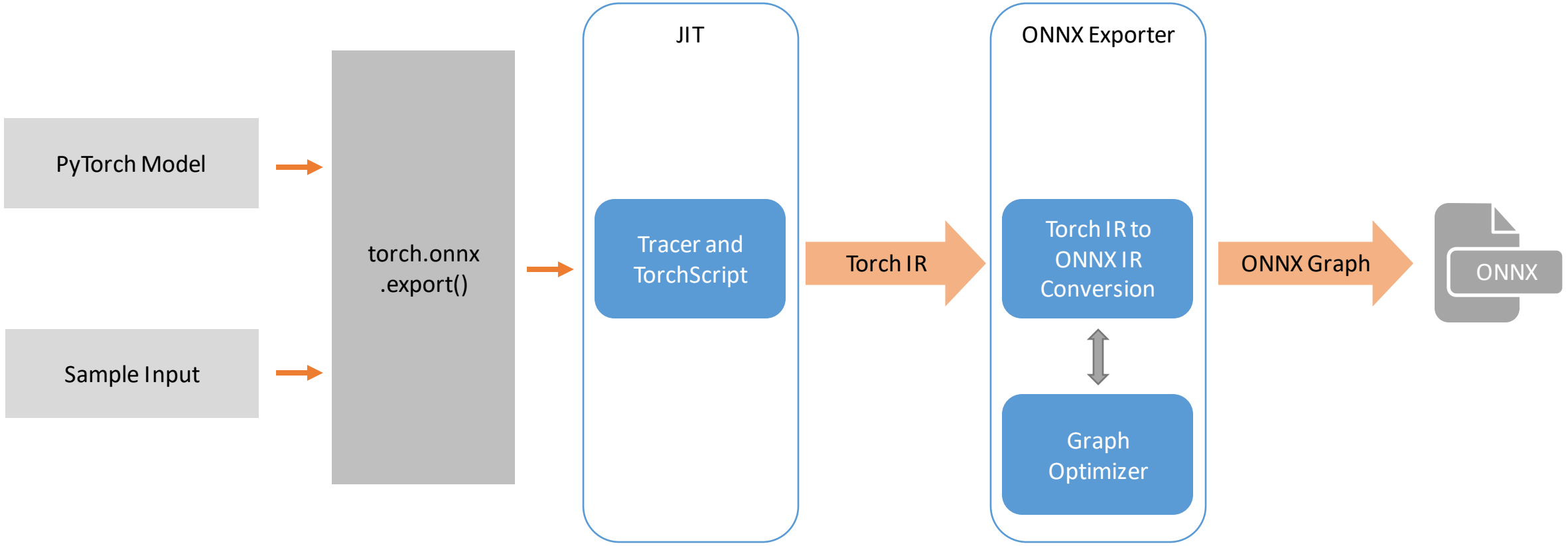
PyTorch Converter for ONNX

Bowen Bao

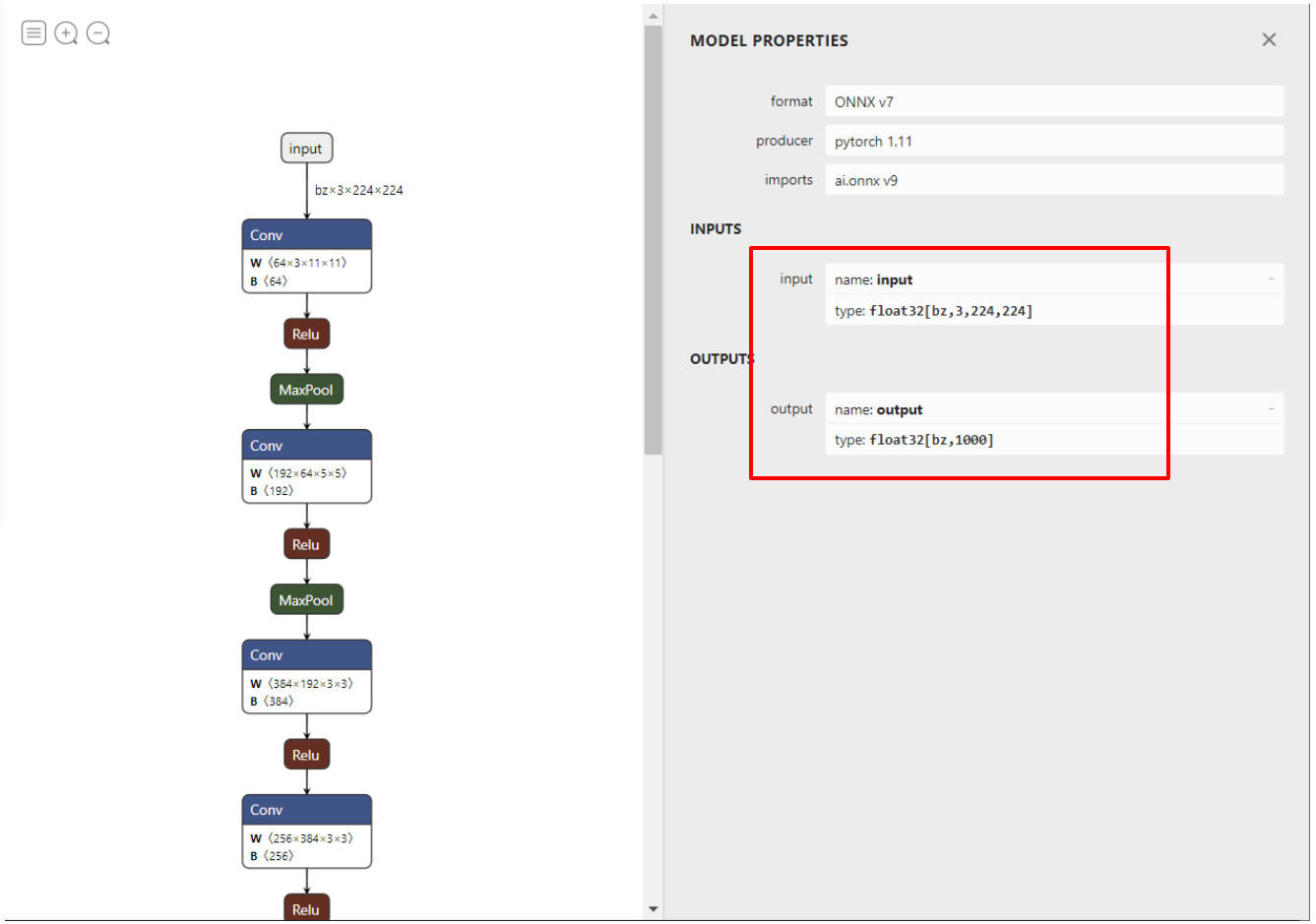
Architecture and Flow



https://github.com/pytorch/pytorch/blob/master/torch/csrc/jit/OVERVIEW.md
https://pytorch.org/docs/stable/onnx.html#torch-onnx
https://github.com/pytorch/pytorch/wiki/PyTorch-ONNX-exporter

End-to-end AlexNet from PyTorch to ONNX

```python
import torch
import torchvision

dummy_input = torch.randn(10, 3, 224, 224)
model = torchvision.models.alexnet(pretrained=True)

torch.onnx.export(model, dummy_input, "alexnet.onnx",
                  input_names=['input'],
                  output_names=['output'],
                  dynamic_axes={
                    'input': {0: 'bz'},
                    'output': {0: 'bz'}})
```

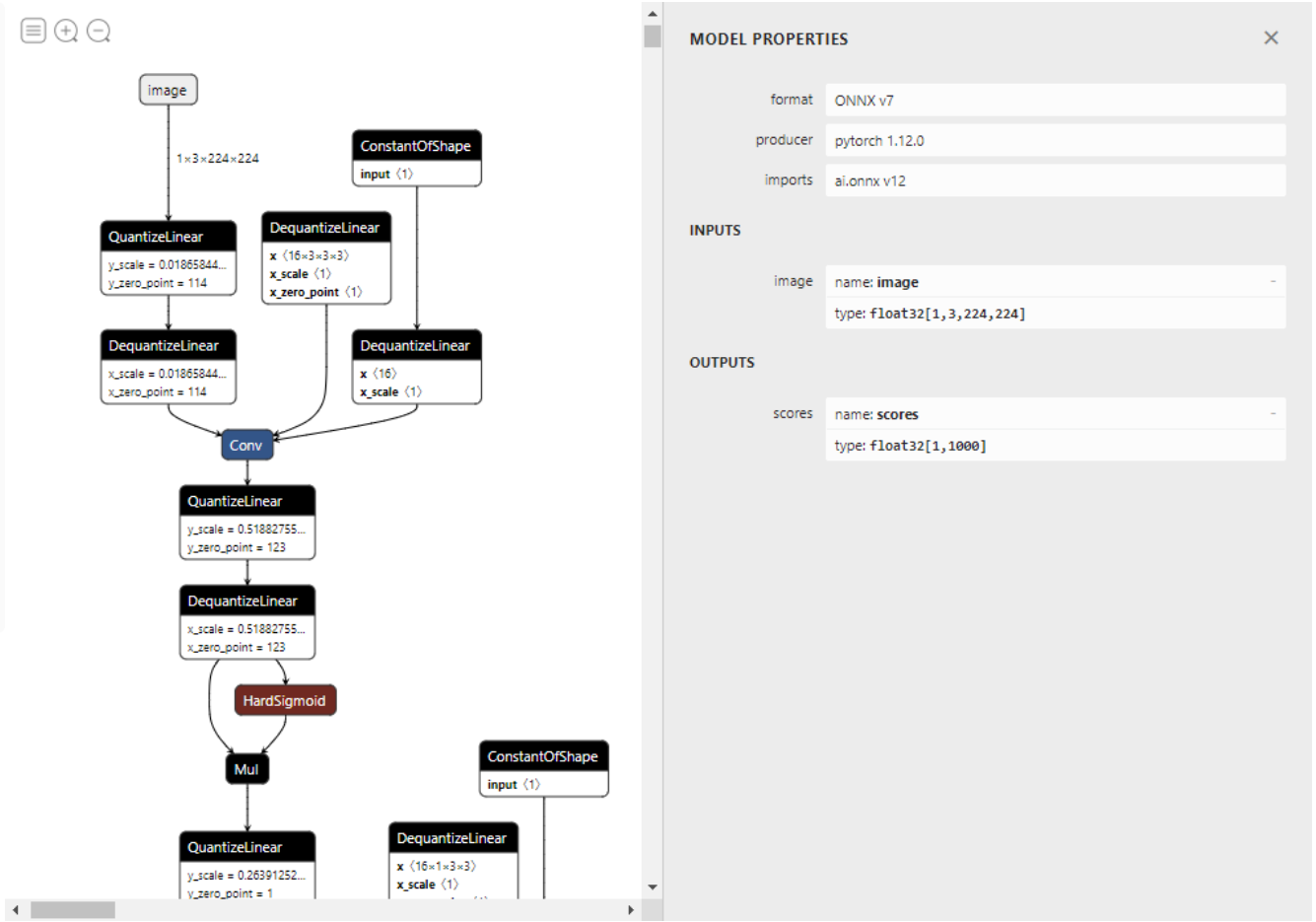


https://pytorch.org/docs/stable/onnx.html#example-alexnet-from-pytorch-to-onnx
Netron for model visualization: https://netron.app/.

Interesting Features

Quantization

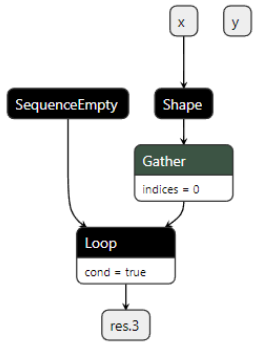
```python
from torchvision.models.quantization import mobilenet_v3_large

model = mobilenet_v3_large(pretrained=True, quantize=True)
model.eval()

input_image = Image.open("dog.jpg")
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225]),
])
input_tensor = preprocess(input_image)
# create a mini-batch as expected by the model
input_batch = input_tensor.unsqueeze(0)

torch.onnx.export(model, input_batch, 'mobilenet_v3_large_quant.onnx',
                  input_names=['image'], output_names=['scores'],
                  opset_version=12,)
```

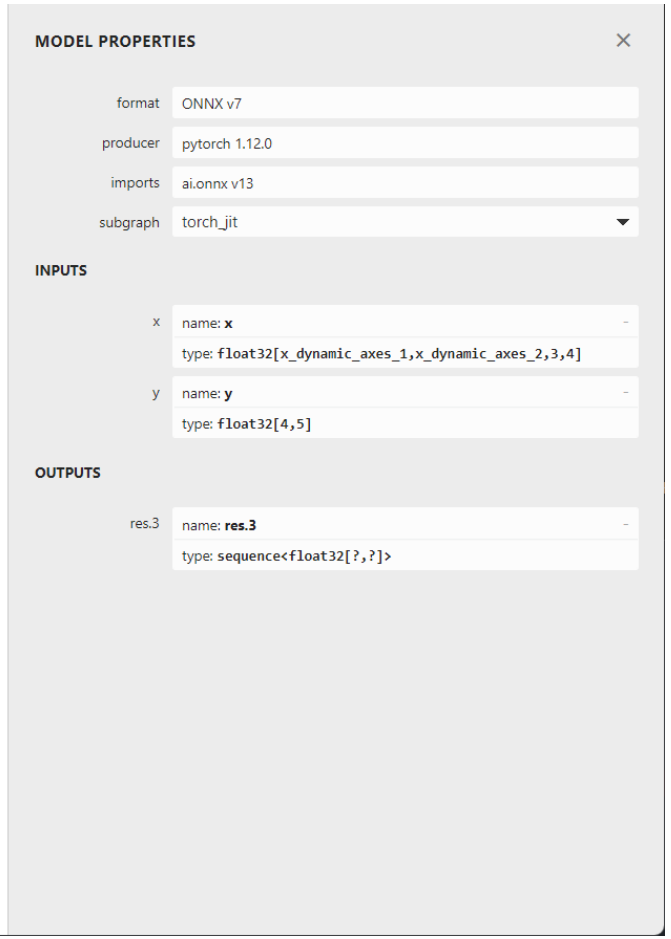


https://pytorch.org/docs/stable/quantization.html
https://github.com/pytorch/pytorch/wiki/PyTorch-ONNX-exporter#quantized-model-export (WIP)

Control Flow

```python
import torch

class Model(torch.nn.Module):
    def forward(self, x, y):
        res = []
        for i in range(x.size(0)):
            for j in range(x.size(1)):
                res += [torch.matmul(x[i][j], y)]
        return res

model = torch.jit.script(Model())
x = torch.randn(4, 4, 3, 4)
y = torch.randn(4, 5)
torch.onnx.export(model, (x, y), "controlflow.onnx",
                  input_names=["x", "y"],
                  dynamic_axes={"x": [0, 1]})
```





https://pytorch.org/docs/stable/onnx.html#tracing-vs-scripting

E.g. Beam Search with Transformers (https://github.com/huggingface/transformers/tree/main/examples/research_projects/onnx/summarization)

Custom ops

- Custom torch.autograd.Function

```
import torch

class MyReLU(torch.autograd.Function):

    @staticmethod
    def forward(ctx, input):
        return input.clamp(min=0)

    @staticmethod
    def symbolic(g, input):
        return g.op("com.microsoft::MyReLU", input)

relu = MyReLU.apply
class Model(torch.nn.Module):
    def forward(self, x):
        return relu(x)
```

https://pytorch.org/docs/stable/onnx.html#torch-autograd-functions

Custom ops

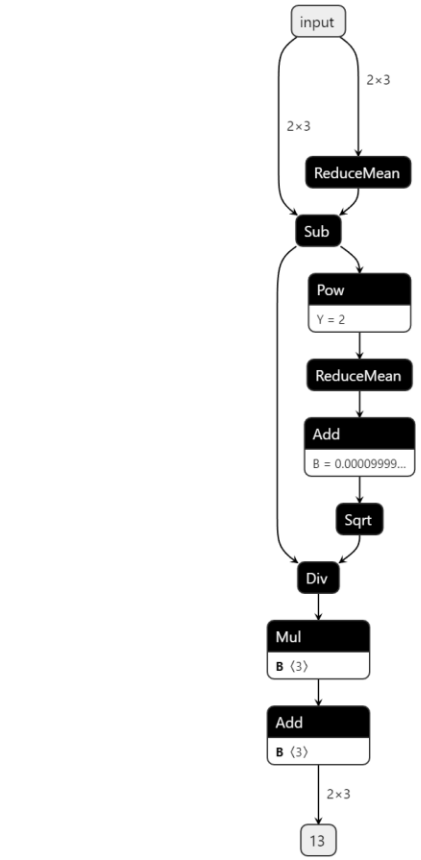
- Register custom symbolic function

```
def symbolic_custom_relu(g, self):
    return g.op("custom_domain::Relu", self)

from torch.onnx import register_custom_op_symbolic
register_custom_op_symbolic("aten::relu", symbolic_custom_relu, opset_version=1)
```

https://pytorch.org/docs/stable/onnx.html#custom-operators

ONNX Local Function

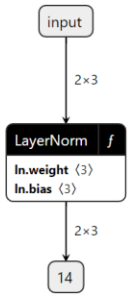
```python
import torch


class M(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.ln = torch.nn.LayerNorm(3, eps=1e-4)

    def forward(self, x):
        return self.ln(x)

x = torch.randn(2, 3)

torch.onnx.export(
    M(),
    (x, ),
    "layernorm.onnx",
    opset_version=16,
)
```

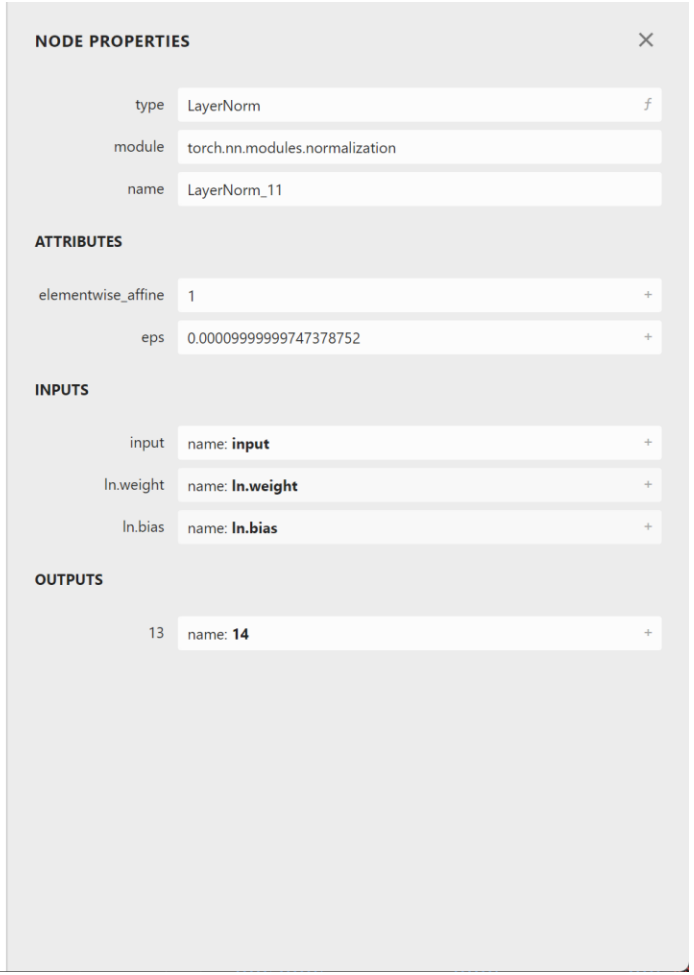


https://pytorch.org/docs/stable/onnx.html#torch.onnx.export
https://github.com/onnx/onnx/blob/main/docs/IR.md#functions

ONNX Local Function

```python
import torch


class M(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.ln = torch.nn.LayerNorm(3, eps=1e-4)

    def forward(self, x):
        return self.ln(x)

x = torch.randn(2, 3)

torch.onnx.export(
    M(),
    (x, ),
    "local_function.onnx",
    export_modules_as_functions={
        torch.nn.LayerNorm,
    },
    opset_version=16,
)
```

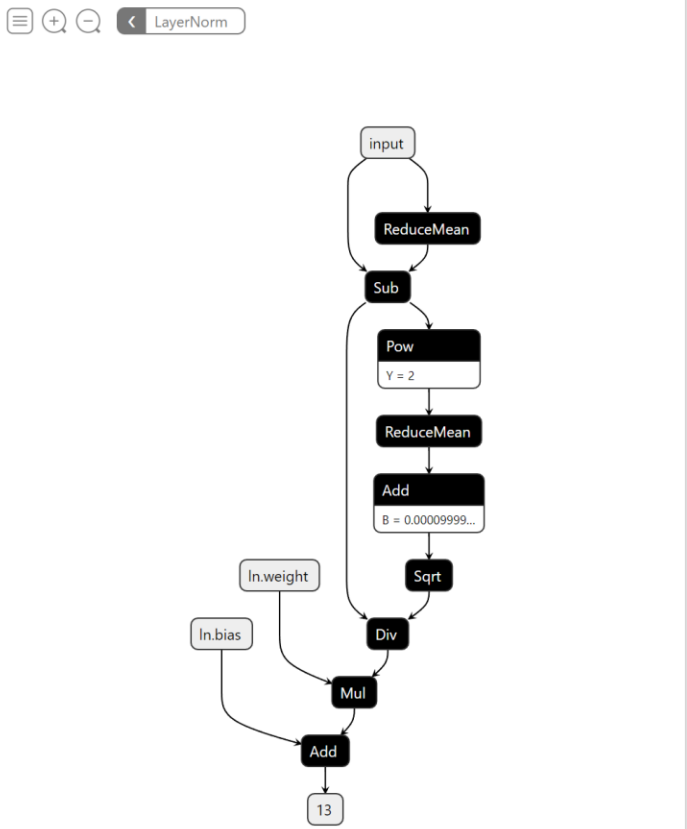


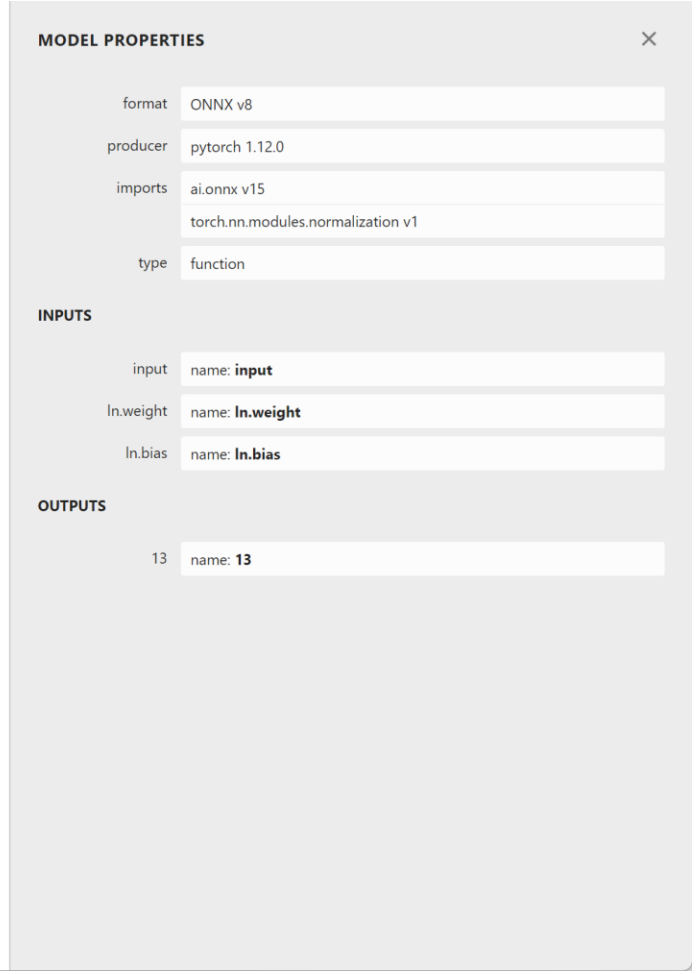


https://pytorch.org/docs/stable/onnx.html#torch.onnx.export
https://github.com/onnx/onnx/blob/main/docs/IR.md#functions

ONNX Local Function

```python
import torch


class M(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.ln = torch.nn.LayerNorm(3, eps=1e-4)

    def forward(self, x):
        return self.ln(x)

x = torch.randn(2, 3)

torch.onnx.export(
    M(),
    (x, ),
    "local_function.onnx",
    export_modules_as_functions={
        torch.nn.LayerNorm,
    },
    opset_version=16,
)
```





https://pytorch.org/docs/stable/onnx.html#torch.onnx.export
https://github.com/onnx/onnx/blob/main/docs/IR.md#functions

Mixed Precision

- torch.autocast
- apex.amp

Roadmap

- Workflow for fixing and diagnosing failures.
- Support more models out of the box.

Useful links

- PyTorch ONNX docs: https://pytorch.org/docs/stable/onnx.html#torch-onnx.
- Nightly version: https://pytorch.org/docs/master/onnx.html.
- Developer docs: https://github.com/pytorch/pytorch/wiki/PyTorch-ONNX-exporter.
- Supported Ops: https://pytorch.org/docs/master/onnx_supported_aten_ops.html.
- PyTorch JIT Overview: https://github.com/pytorch/pytorch/blob/master/torch/csrc/jit/OVERVIEW.md.
- Netron for model visualization: https://netron.app/.
- File issues under PyTorch: https://github.com/pytorch/pytorch/issues/new/choose.

Thank you!