What's new in torch.onnx

Justin Chu, Maanav Dalal, Xavier Dupré, Jambay Kinley, Ganesan Ramalingam, Kunal Vaishnavi, Ti-Tai Wang

Microsoft

Agenda

- A refresher on ONNX and torch.onnx
- What is new in torch.onnx
  - New export API and torch.export based architecture
  - Control flow support with torch.cond
  - Torch native ONNX operators for influencing the tracer
  - ONNXProgram API
  - Metadata and validation tools
- Examples and Demo
- Tips for successful export

A refresher on ONNX and torch.onnx

- Open Neural Network Exchange (ONNX) is an open standard for AI models
- Recent efforts have been focused around representing GenAI model efficiently
- torch.onnx enables conversion from PyTorch to ONNX

The new ONNX exporter: torch.export based architecture

- Use torch.export to capture the FX graph
  - Replaces TorchScript
  - Low memory usage
- New set of aten->onnx conversion logic (“torchlib” in ONNX Script) with full support dynamic shapes + ONNX Opset 18-24
- Efficient graph construction using ONNX Script and ONNX IR

New API

- The dynamo=True option is default starting from **torch 2.9**

- Returns ONNXProgram

New API Sample

```
torch.onnx.export(
    model,
    args,
    kwargs=kwargs,
    dynamic_shapes=dynamic_shapes,
    [dynamic_axes=]
    [dynamo=True,]
    report=True,
    verify=True
) -> torch.onnx.ONNXProgram
```

Control Flow support in PyTorch 2

- torch.cond, torch.scan support in PyTorch 2.9; torch.while_loop to be supported in 2.10
- Useful when: Need any control flow logic that depend on the tensor data

Conditional Operation

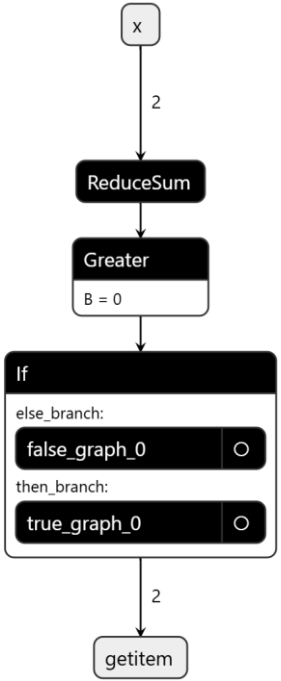
```
class Model(torch.nn.Module):
    def forward(self, x):
        if x.sum():
            return x * 2
        return -x
```

Rewritten

```
class Model(torch.nn.Module):
    def forward(self, x):
        def times_2(x):
            return x * 2

        def neg(x):
            return -x

        return torch.cond(
            x.sum() > 0,
            times_2,
            neg,
            (x,)
        )
```

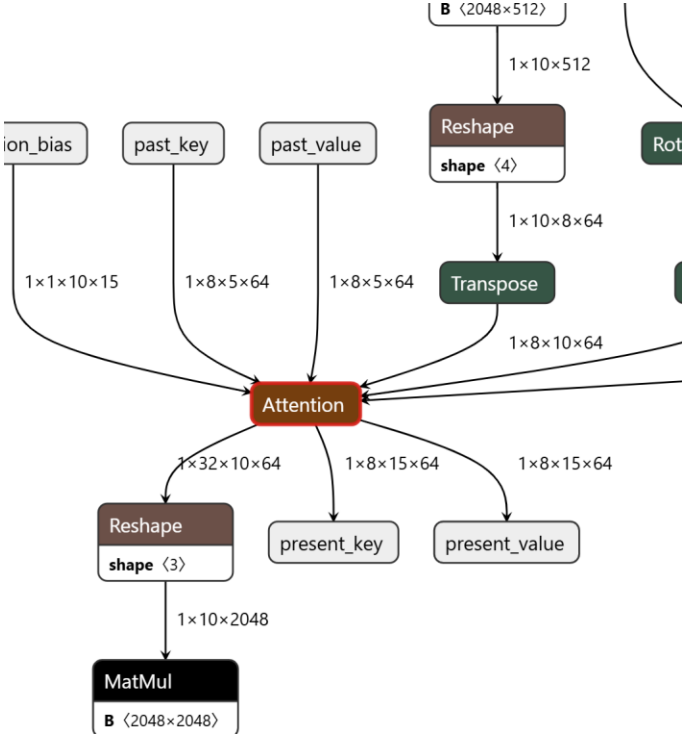


PyTorch native ONNX operator

Attention Op

```
def forward(
        self,
        hidden_states: torch.Tensor,
        attention_mask: torch.Tensor,
        position_ids: torch.Tensor,
        rope_cos_freqs: torch.Tensor,
        rope_sin_freqs: torch.Tensor,
        past_key: torch.Tensor,
        past_value: torch.Tensor,
    ) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
        ...

        attention_output, present_key, present_value, _ =
torch.onnx.ops.attention(
            query_states,
            key_states,
            value_states,
            full_mask,
            past_key,
            past_vale,
            kv_num_heads=self.num_key_value_heads,
            q_num_heads=self.num_attention_heads,
            scale=self.scaling,
        )
```



PyTorch native ONNX operator – Custom Ops

- Use torch.onnx.is_in_onnx_export() to guard the ONNX export logic and place it next to your normal model forward code
- torch.onnx.ops.symbolic()
- torch.onnx.ops.symbolic_multi_out()

- Useful when: you want to insert custom ops or influence the graph created by the exporter

Custom MatMulNBits

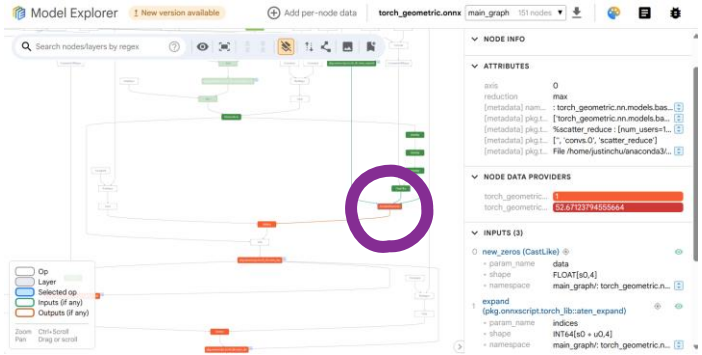
```
def forward(self, ...):
    if torch.onnx.is_in_onnx_export():
        return torch.onnx.ops.symbolic(
            "com.microsoft::MatMulNBits",
            [x, qweight, scales, qzeros],
            attrs={
                "K": in_features,
                "N": out_features,
                "bits": bits,
                "block_size": group_size,
                "accuracy_level": 4,
            },
            dtype=x.dtype,
            shape=[*x.shape[:-1], out_features],
            version=1,
        )
    else:
        ...
```

ONNXProgram

- Model stored and **optimized in ONNX IR** (control with optimize=True)
  - Pattern based fusion
  - Constant propagation
  - Weight deduplication
  - Dead code elimination
  - Common subexpression elimination
  - …
- Can **run directly on torch.Tensor** inputs with ONNX Runtime for model validation
- Further **graph manipulation** with onnx_program.model using onnx-ir
- When done, **serialize efficiently** to onnx file with onnx_program.save()

Validation Tools

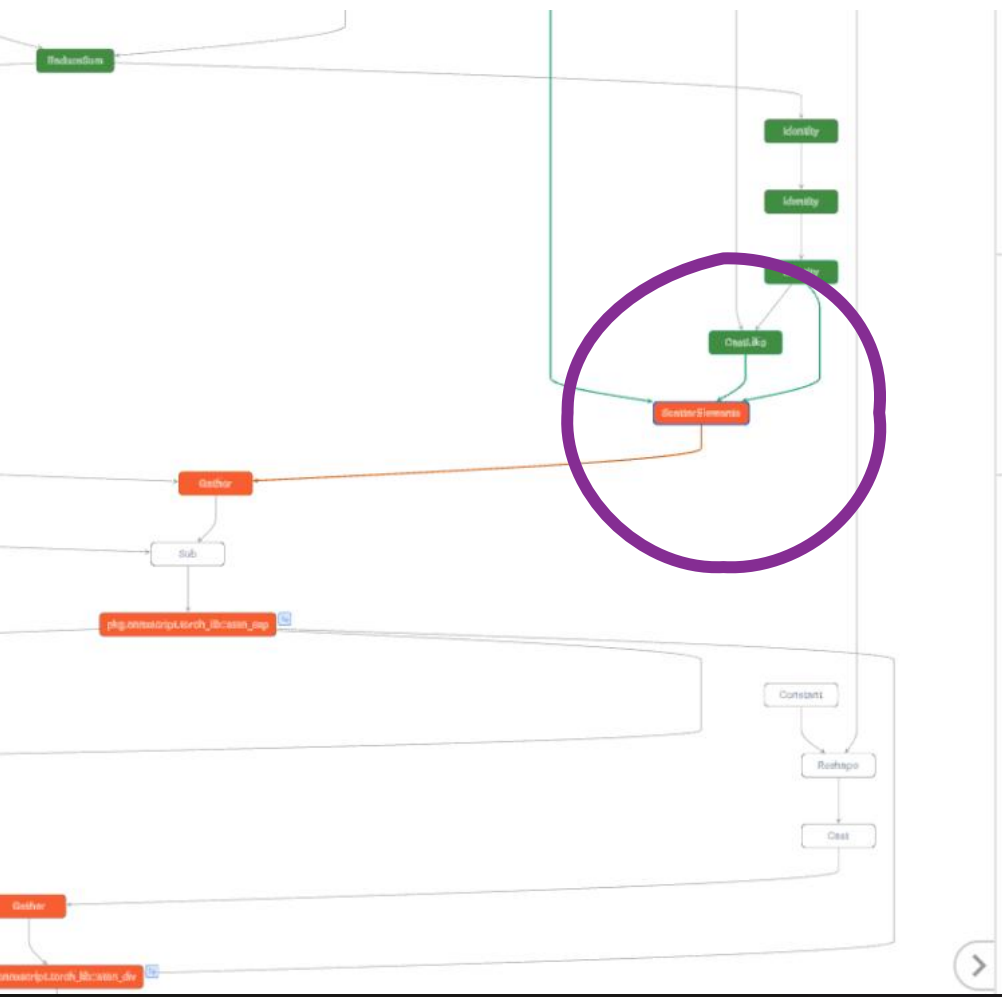
torch.onnx.verification.verify_onnx_program for diagnosing model accuracy by comparing intermediate tensors with the ExportedProgram



Verifying ONNX Program

```
import torch
from torch.onnx.verification import verify_onnx_program

from model_explorer_onnx.torch_utils import
save_node_data_from_verification_info
# Export the and save model
onnx_program = torch.onnx.export(model, args, dynamo=True)
onnx_program.save("model.onnx")
verification_infos = verify_onnx_program(onnx_program,
compare_intermediates=True)
# Produce node data for Model Explorer for visualization
save_node_data_from_verification_info(
    verification_infos, onnx_program.model, model_name="model"
)
```



[metadata] nam... : torch_geometric.nn.models.bas...
[metadata] pkg.t... ['torch_geometric.nn.models.ba...
[metadata] pkg.t... %scatter_reduce : [num_users=1...
[metadata] pkg.t... ['', 'convs.0', 'scatter_reduce']
[metadata] pkg.t... File /home/justinchu/anaconda3/...

NODE DATA PROVIDERS

torch_geometric... 1
torch_geometric... 52.67123794555664

INPUTS (3)

0 new_zeros (CastLike)
- param_name data
- shape FLOAT[s0,4]
- namespace main_graph/: torch_geometric.n...

1 expand (pkg.onnxscript.torch_lib::aten_expand)
- param_name indices
- shape INT64[s0 + u0,4]
- namespace main_graph/: torch_geometric.n...

Demo
Foundry Local

Tips for successful export

- Wrap model with nn.module that takes plain torch.Tensor inputs
- Preserve dynamism of the model
  - Use torch.cond, torch.scan for control flows
  - Use .shape[] instead of len() to preserve dynamic shapes
  - Use torch._check instead of assert on dynamic shapes
- Use torch.onnx.ops.symbolic to create custom ops or to bypass non-tracible sections
- Refer to ExportDB: https://docs.pytorch.org/docs/stable/generated/exportdb/index.html

- Use torch nightly

Thanks!

Justin Chu, Maanav Dalal
Microsoft

ONNX Docs

Demo Repo

FoundryLocal.ai