



CONFERENCE

—NORTH AMERICA 2026—

Lightweight FX Tracing in PyTorch

Yidi Wu, Richard Zou

Computational Graph

```
def deepseek_v3_transformer_forward(...):  
    attn_out = self.attn(self.attn_norm(x), attn_meta,  
        pos)  
    x = x + attn_out  
    if self.moe_enabled:  
        ffn_out = self.moe(self.ffn_norm(x))  
    else:  
        ffn_out = self.feed_forward(self.ffn_norm(x))  
    return x + ffn_out
```

```
def fx_forward(...):  
    attn_out = ...  
    x = torch.ops.aten.add.Tensor(x, attn_out)  
    moe_input, rstd = torch.ops.aten._fused_rms_norm.default(  
        x, [7168], ffn_norm_weight, 1e-5  
    )  
  
    ffn_out = ... # MoE.forward scope  
    out = torch.ops.aten.add.Tensor(x, ffn_out)  
    return out
```

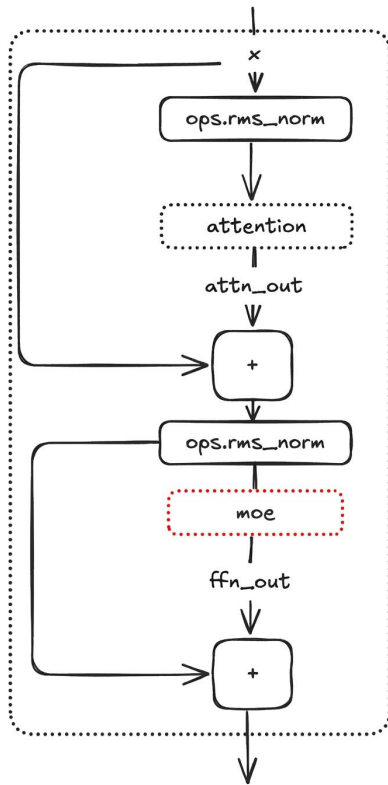


Figure 2: an example deepseek_v3 moe graph.

Key Benefits

Identify Optimizations

- Fusion opportunities
- Implicit casts and copies
- Critical path analysis

Share Optimizations

- Different models
- Heterogeneous hardware

Integrate Optimizations

- Aggressive fusions that cross semantic boundaries
- Agent hill-climbing

Graph Capture: Options and Tradeoff

- dynamo:
 - Pros: captures orchestration code
 - Cons: occasional coverage gaps, graph breaks
- make_fx (`__torch_dispatch__` based):
 - Pros: reliably captures full operator graph
 - Cons: can't capture orchestration code
- Coding Agents:
 - Pros: flexible and easy to use
 - Cons: non-deterministic behavior, token cost

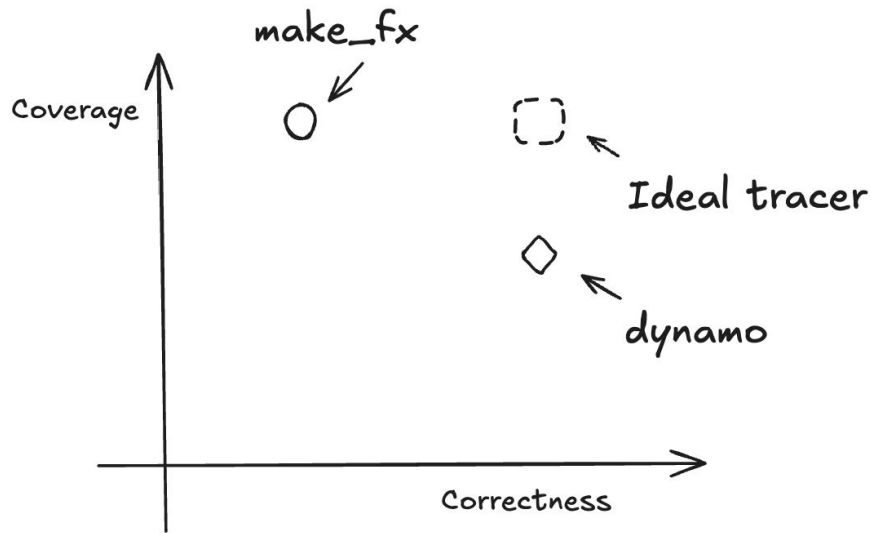


Figure 2: correctness vs coverage difference between dynamo and make_fx

Correctness

Key observations: if the model is CUDA-Graphable, the tracer only needs to faithfully map the tape of kernels to graph nodes.

CUDA Graph:

- CUDA Graph are essential for models running on modern GPUs. It removes cpu launch overhead by invoking kernels on device.
- At CUDA Graph capture stage, the program is captured as a tape of CUDA kernels.
- When replaying the CUDA Graph, only the recorded kernels are replayed.

A PyTorch program is CUDA-Graphable if we could replace it with the captured CUDA Graph replay.

Best of Both Worlds: cudagraph_fx

APIs:

- `trace = make_cudagraph_fx(fn)(*args, **kwargs)`
- `trace.replay(*args, **kwargs)`

Mental Correctness Model:

- Kernels invoked by PyTorch operators are captured automatically
- Kernels that don't go through PyTorch operators will be reported as error with clear stacktrace. The fix is to wrap them as custom operators.
- Streams and Events are also carried in the graph.

Case Study: Fusion Explorer

A quick [demo](#) for a vibe coded visualization of TorchTitan deepseek v3.

Takeaways

- ``cudagraph_fx`` brings lightweight FX tracing to CUDA-Graphable workloads.
- Kernel coverage is checked; streams and events are preserved.
- FX graphs help identify, apply, and share optimizations.
- Try at https://github.com/meta-pytorch/torch_cudagraph_fx!