PyTorch CONFERENCE 2025

Efficient INT8 Inference on ARM: Leveraging PyTorch 2 Export Quantization

N Maajid Khan, Devang Choudhary, Abhishek Jain
Fujitsu Research of India Pvt. Ltd.

FUJITSU

Fujitsu Monaka

Abstract

With the rapid adoption of ARM CPUs across cloud and edge environments, high-performance quantized inference in PyTorch on ARM is now critical for research and production. Previously, PT2E inference used only FP32 kernels, limiting acceleration and efficiency. We introduce enhancements to the **PyTorch 2 Export Quantization (PT2E)** workflow, unlocking efficient INT8 inference for ARM targets.

Our contributions include:
• **ARM Backend Integration**: Extending the PT2E quantization path from x86 to ARM CPUs, leveraging oneDNN JIT and Arm Compute Library (ACL) INT8 kernels for operators such as matrix multiplication and convolution.[1]

• **Quantization Granularity & API**: Introduced "per_channel" quantized weights (with "per_tensor" planned) and updated the API for easy granularity selection. Supports static and dynamic quantization, with seamless integration of Quantization Aware Training (QAT) and Post-Training Quantization (PTQ).

• Comprehensive Model Support: Validated on NLP (BERT, T5), vision (ResNet, ViT), and custom models, demonstrating broad applicability.

These improvements enable out-of-the-box INT8 inference on ARM platforms for scalable, efficient, portable AI deployment.

Keywords: PyTorch 2 Export (PT2E), ARM CPUs, INT8 Inference, oneDNN.

Methodology

We extend the PyTorch 2 Export (PT2E) quantization stack to enable high-performance INT8 inference on ARM CPUs. PT2E consolidates quantization support by phasing out legacy pathways (Eager Mode, TorchScript Graph Mode, FX Graph Mode) into a single unified stack. The PT2E implementation has also been migrated to torchao, a standalone library decoupled from PyTorch core, enabling shared reuse of observers, fake quantizers, and other components.

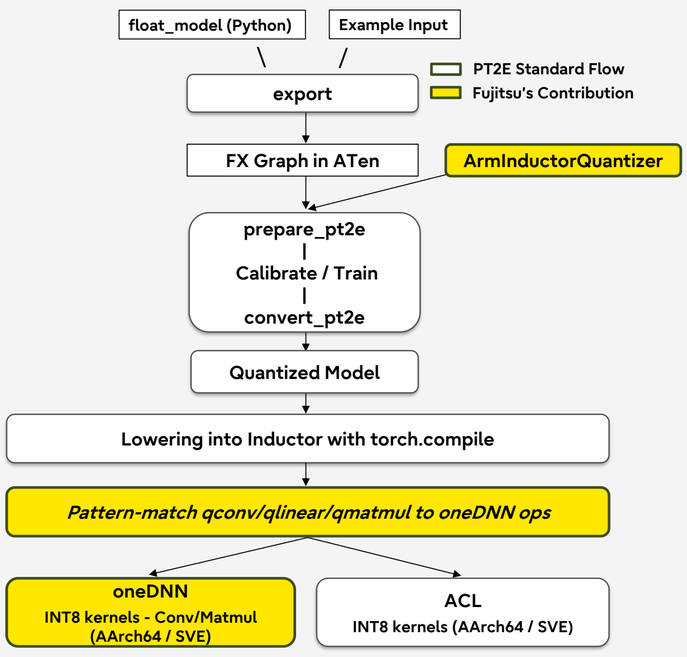

Figure 1 : PT2E quantization workflow extended for ARM CPUs.

As shown in **Figure 1**, the workflow begins with a floating-point model exported to an FX Graph in ATen. Fujitsu's key contribution, highlighted in yellow, is the **ArmInductorQuantizer**, which:

- Applies quantization annotations and recipes during the prepare_pt2e stage.
- Provides default configurations for convolution, linear, and matmul.
- Supports both Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT).
- Detects common fusion patterns (e.g., conv+bn, linear+unary) to produce conversion-ready graphs.

After convert_pt2e, we extend torch.compile (Inductor) with ARM-specific lowering. Quantized ops (qconv, qlinear, qmatmul) are pattern-matched and lowered to oneDNN primitives, including JIT and Arm Compute Library (ACL) INT8 kernels.

To further enhance the INT8 path, we added new oneDNN JIT INT8 kernels. [2]

- Matmul: Quantized INT8 kernel
- Convolution: BRGEMM with u8:s8:f32 support
- Convolution: BRGEMM with u8:s8:u8 support

These enhancements, validated on NLP (BERT, T5), vision (ResNet, ViT), and custom models, deliver up to 2.1x performance gains compared to FP32 while preserving accuracy, enabling efficient and scalable INT8 inference on ARM.

Results

We evaluated the INT8 PT2E workflow on AWS Graviton3E (AArch64, 32 cores) system with **PyTorch v2.7.1, oneDNN v3.7.1, and ACL v52.0.1.** Quantization was tested using both Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT), across static and dynamic modes, with per-channel weight quantization enabled.

Representative models from vision (ResNet50, ViT) and NLP (BERT, T5) were inferenced using the PT2E API and compiled with Inductor for ARM backends. Inference was run with tcmalloc enabled and channels-last settings for vision workloads, using average latency per input (ms) as the metric.

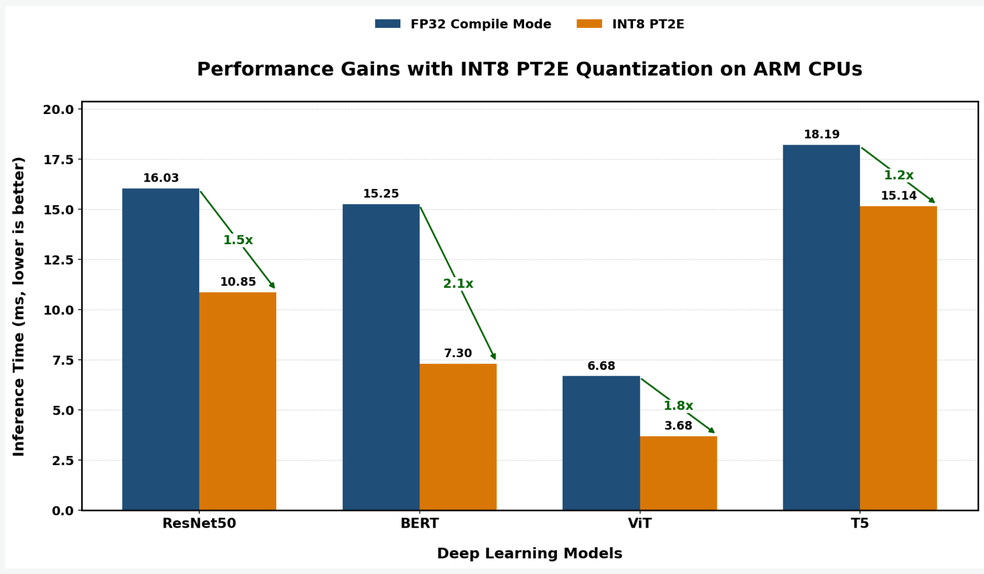

Figure 2: Performance Gains with INT8 PT2E Quantization on ARM CPUs

Figure 2 demonstrates the speed-up of INT8 PT2E inference compared to FP32 + compile on an AWS Graviton3E. The reduction in bar height across models highlights the latency savings.

Results show consistent performance improvements across representative NLP and vision models: **BERT (2.1x), ResNet50 (1.5x), ViT (1.8x), and T5 (1.2x)**. On average, INT8 PT2E reduced inference latency by up to 2.1x while maintaining accuracy.

These results validate our ARM backend integration as an important step toward efficient, scalable, and production-ready INT8 quantization on ARM CPUs.

Conclusion and Future Work

We extend the PT2E quantization stack to enable efficient INT8 inference on ARM CPUs, aligning with PyTorch's long-term roadmap of consolidating quantization under PT2E. With the ArmInductorQuantizer and ARM-specific Inductor lowering, we demonstrate speedups up to 2.1x over FP32 across NLP and vision models, validating the effectiveness of our contributions.

Looking ahead, we plan to enhance the lowering pass, broaden operator coverage and contribute additional oneDNN JIT-based INT8 kernels.

We will also introduce new fusion patterns for fused operator execution to further accelerate inference, ensuring robust and production-ready quantization workflows for ARM within the PyTorch ecosystem.

References

[1] PyTorch 2 Export Quantization.
https://docs.pytorch.org/docs/stable/quantization.html#prototype-pytorch-2-export-quantization

[2] Scalable Vector Extensions, Arm Developer.
https://developer.arm.com/Architectures/Scalable%20Vector%20Extensions

Links to pull requests

ARM Backend Integration

Quantization Granularity & API

Matmul oneDNN INT8

Convolution oneDNN INT8 with u8:s8:f32

Convolution oneDNN INT8 with u8:s8:u8

This poster is based on results obtained from a project, JPNP21029 subsidized by the New energy and Industrial Technology Development Organization (NEDO).