ExecuTorch

A Unified PyTorch Solution to Run AI Models On-Device

Mergen Nachin, Digant Desai, Sicheng Stephen Jia, Chen Lai, Mengwei Liu, Jacob Szwejbka, Raziel Alvarez, RJ Ascani, Dave Bort, Manuel Candales, Andrew Caples, Yanan Cao, Zhengxu Chen, Soumith Chintala, Gregory Comer, Tanvir Islam, Songhao Jia, Tarun Karuturi, Jack Khuu, Abhinay Kukkadapu, Tugsbayasgalan Manlaibaatar, Andrew Or, Kimish Patel, Siddartha Pothapragada, Lucy Qiu, Supriya Rao, Orion Reblitz-Richardson, Max Ren, Scott Roy, Anthony Shoumikhin, Scott Wolchok, Guang Yang, Angela Yi, Martin Yuan, Hansong Zhang, Jack Zhang, Jerry Zhang, Shunting Zhang, C. Cagatay Bilgin

Meta

MLSys 2026

Why On-Device AI?

Enhanced Privacy

Data never leaves the device. Process personal content, conversations, and media locally without cloud exposure.

Real-Time Response

Instant inference with no network round-trips. Perfect for AR/VR, autonomous driving, and voice agents.

Connectivity

Works seamlessly in low-bandwidth regions, remote areas, or completely offline. No high-speed network required.

Cost Efficient

No cloud compute bills or API limits. Scale to billions of users without infrastructure costs growing linearly.

Why is On-Device AI hard?

Hardware Heterogeneity

- Varying compute capability and SoC configurations
- Memory constraints (RAM and Storage)
- Bespoke vendor compilers for NPUs

Battery Powered

- Constrained Power and Thermal envelopes
- Limited TOPS and memory bandwidth

Research to Production Gaps

- Numerical differences from model conversions
- Debugging is non-trivial
- Model compression trade-offs for on-device
- Updating and maintaining models

ExecuTorch Guiding Principles

Portability

Support wide variety of computing platforms, from desktop to constrained embedded systems.

Productivity

Enable the same PyTorch experience for authoring, debugging, and deployment across platforms.

Performance

High-performance experience via lightweight runtime utilizing full hardware capabilities.

Community

Develop in the open, encouraging community contributions and feedback

Modularity

Well-defined workflow for program capture, transform, and execution with OOTB components.

Existing Solutions

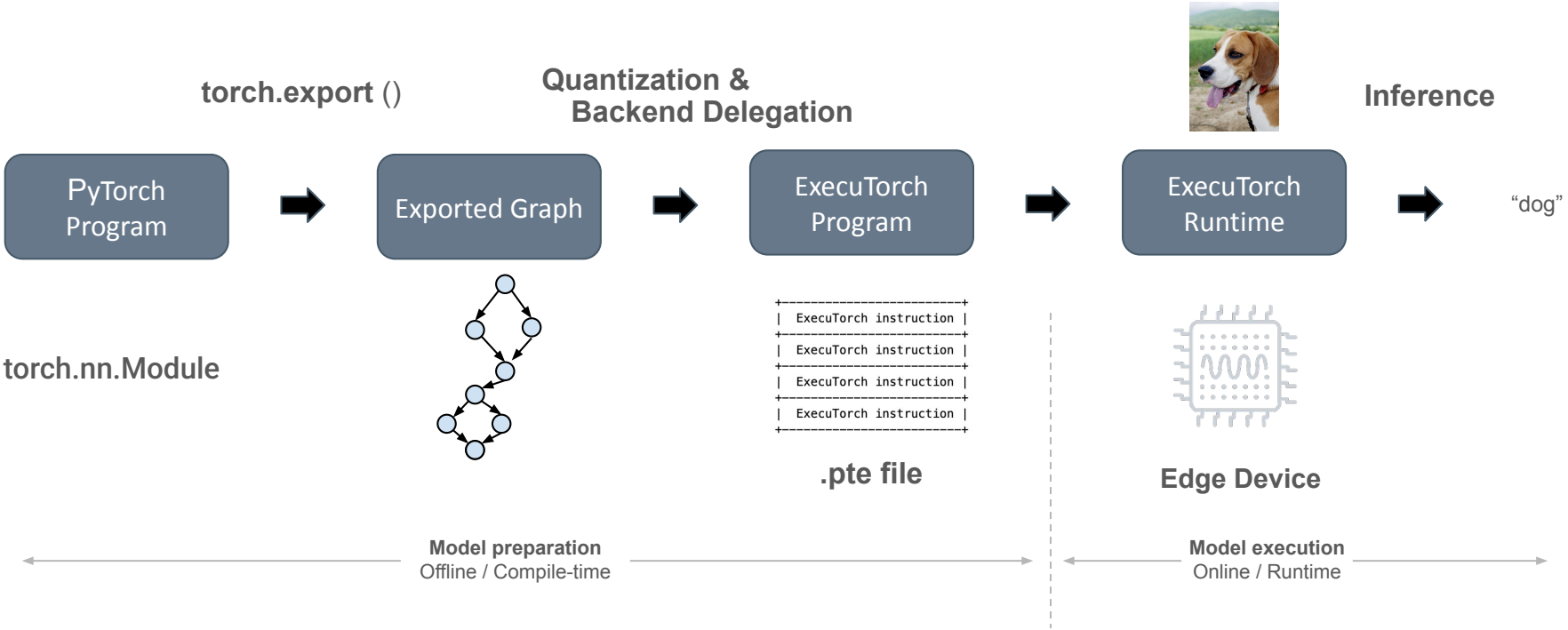
Conversion-based

Compiler-based

Vendor-specific

Model-Specific

ExecuTorch


torch.export ()
Quantization &
Backend Delegation
Inference
PyTorch
Program
Exported Graph
ExecuTorch
Program
ExecuTorch
Runtime
"dog"
torch.nn.Module
+-------------------------+
| ExecuTorch instruction |
+-------------------------+
| ExecuTorch instruction |
+-------------------------+
| ExecuTorch instruction |
+-------------------------+
| ExecuTorch instruction |
+-------------------------+
.pte file
Edge Device
Model preparation
Offline / Compile-time
Model execution
Online / Runtime

Key Architectural Choices

- ➔ Building ExecuTorch on top of PyTorch 2.0 and `torch.export`
- ➔ Introducing Graph based, Backend-aware Quantization
- ➔ Ahead-of-Time, Composable, Optional Backend Delegates
- ➔ Portable and Tiny Runtime with Batteries included

torch.export and ExecuTorch

What does this design choice enable?

- PyTorch powers >90% of AI Research. This lets us leverage it directly
- Backends can work with Core ATen ops (<300), and executable FX graph
- PyTorch tooling and ecosystem support are available

How?

- Take a PyTorch model, export it, and lower it via ExecuTorch*
  - Supports models from Hugging Face via Optimum-ExecuTorch
- Progressive lowering within PyTorch
  - torch.nn.Module → Export IR → Edge IR

* See appendix for torch.export limitations

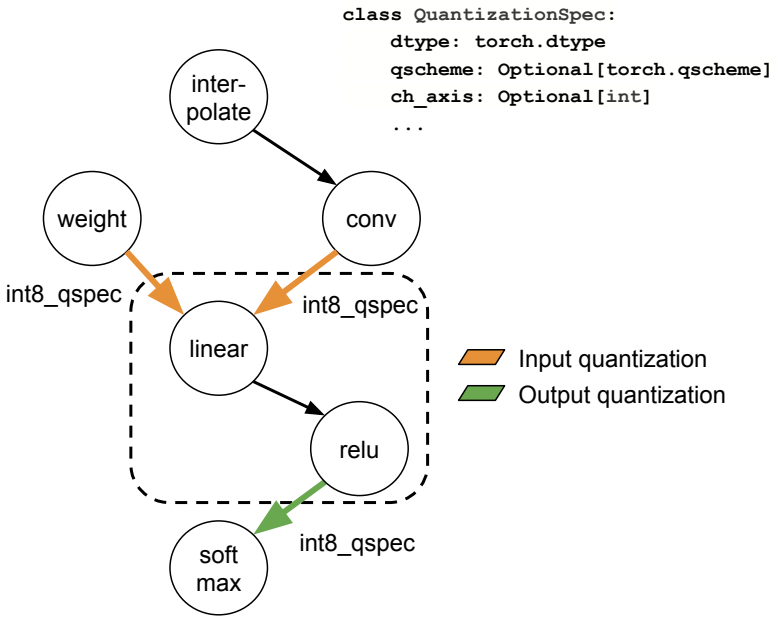
Graph Quantization

What does this design choice enable?

- Hardware specific quantization for backends
- Numerical accuracy tracing at node level
- Same flow from training to deployment
- Seamless evals on server

How?

- Backend specific `Quantizer*`
- Supporting both PTQ and QAT



* ExecuTorch also supports pre-export, eager-based quantization flows through TorchAO.

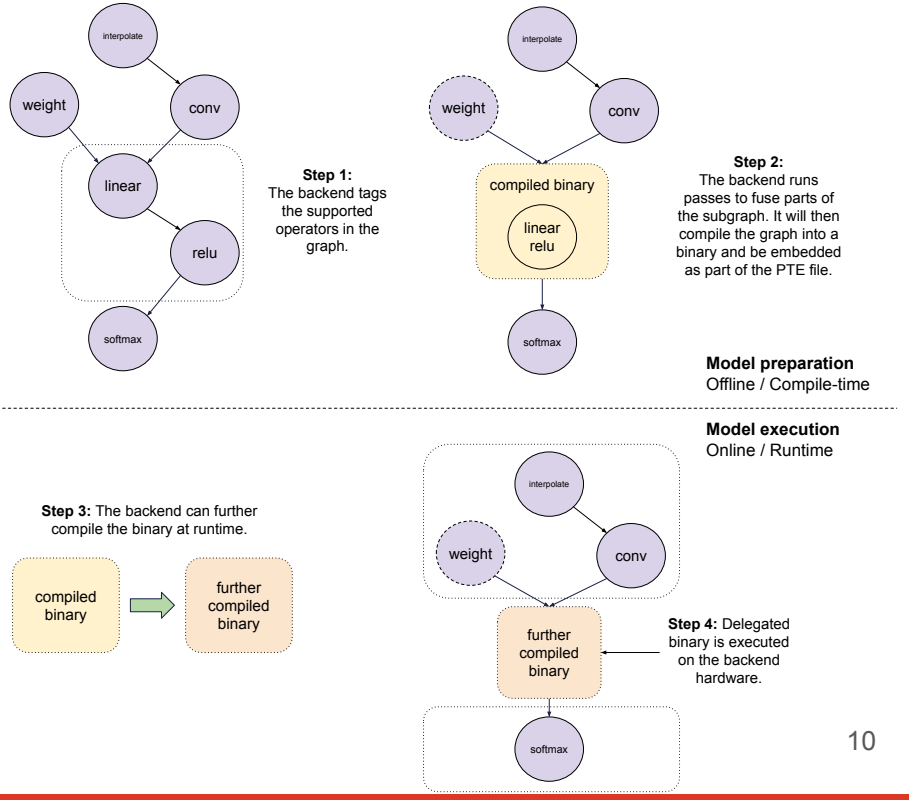
Backend Delegates

What does this design choice enable?

- Abstracts away hardware heterogeneity, uses same IR
- Enables vendor tooling
- Simpler core runtime, making it portable

How?

- Ahead-of-time graph partitioning
- Delegates are composable and optional
- Seamless fall back to Portable kernels



Backend Delegates

CPU

- XNNPACK: Arm, x86 SIMD
- Cortex-M: CMSIS-NN
- Cadence: Hifi, Fusion, Vision DSPs

GPU

- Vulkan: Android GPUs
- Arm VGF: Arm Mali GPUs
- MLX: Apple Silicon GPU
- AOTI Metal: Apple Silicon GPU
- AOTI CUDA: NVIDIA GPU
- TensorRT: NVIDIA GPU

NPU

- Qualcomm QNN: Hexagon HTP
- Arm Ethos-U: Ethos-U55/U85
- Mediatek Neuropilot: Dimensity NPUs
- Samsung ENN: Exynos NPU
- NXP Neutron: eIQ Neutron NPU

Multi-hardware

- CoreML: Apple ANE, GPU, CPU
- OpenVino: Intel NPU, GPU, CPU

Portable Runtime

What does this design choice enable?

- Wide platform support: Hardware, OS

How?

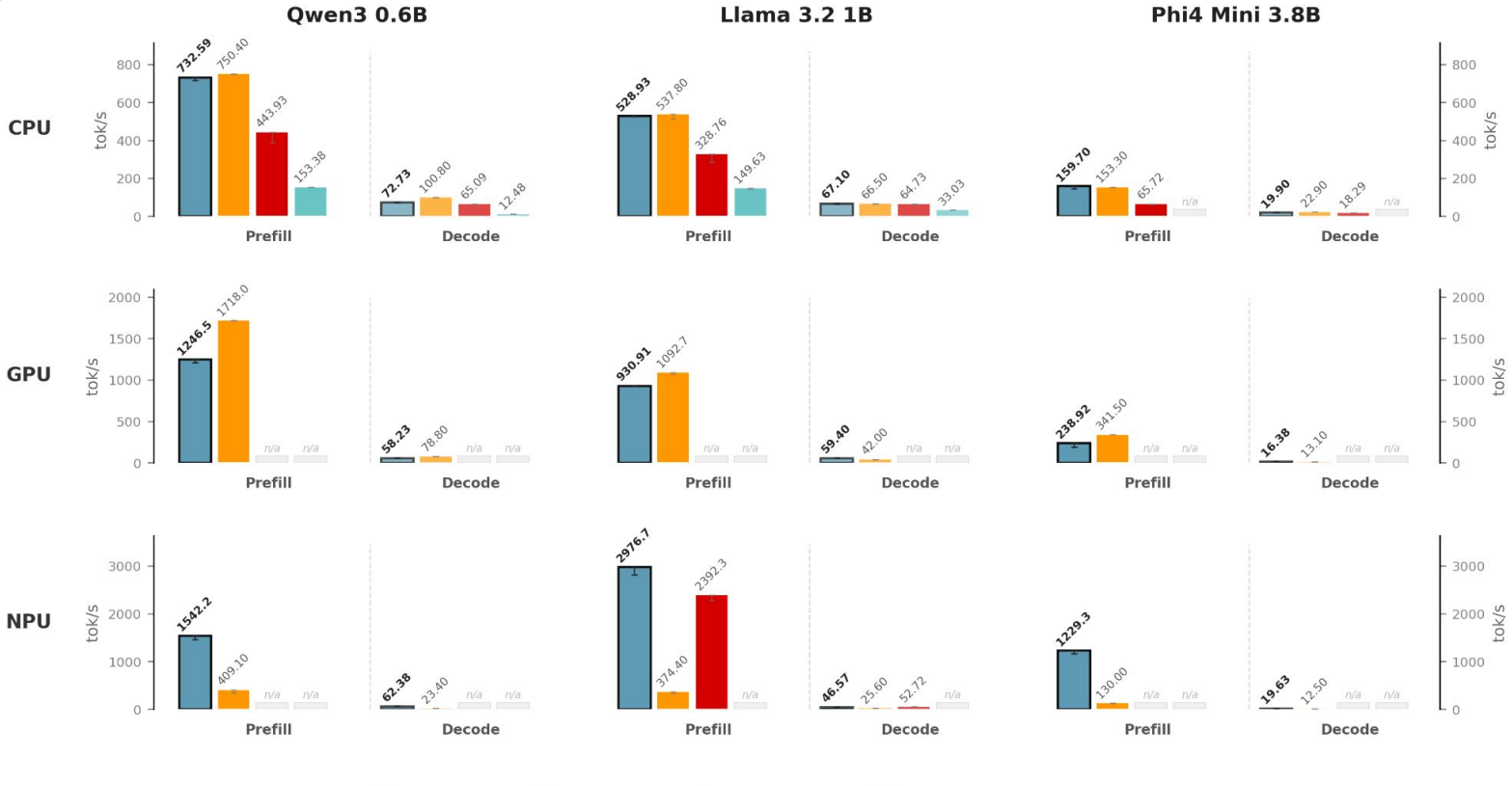
- Light, C++17 based, portable execution engine
- Platform Abstraction Layer for system ops
- Aims to reduce the framework tax
- Built-in Kernels
  - Portable Kernel Library
  - Op and dtype selective build to reduce binary size

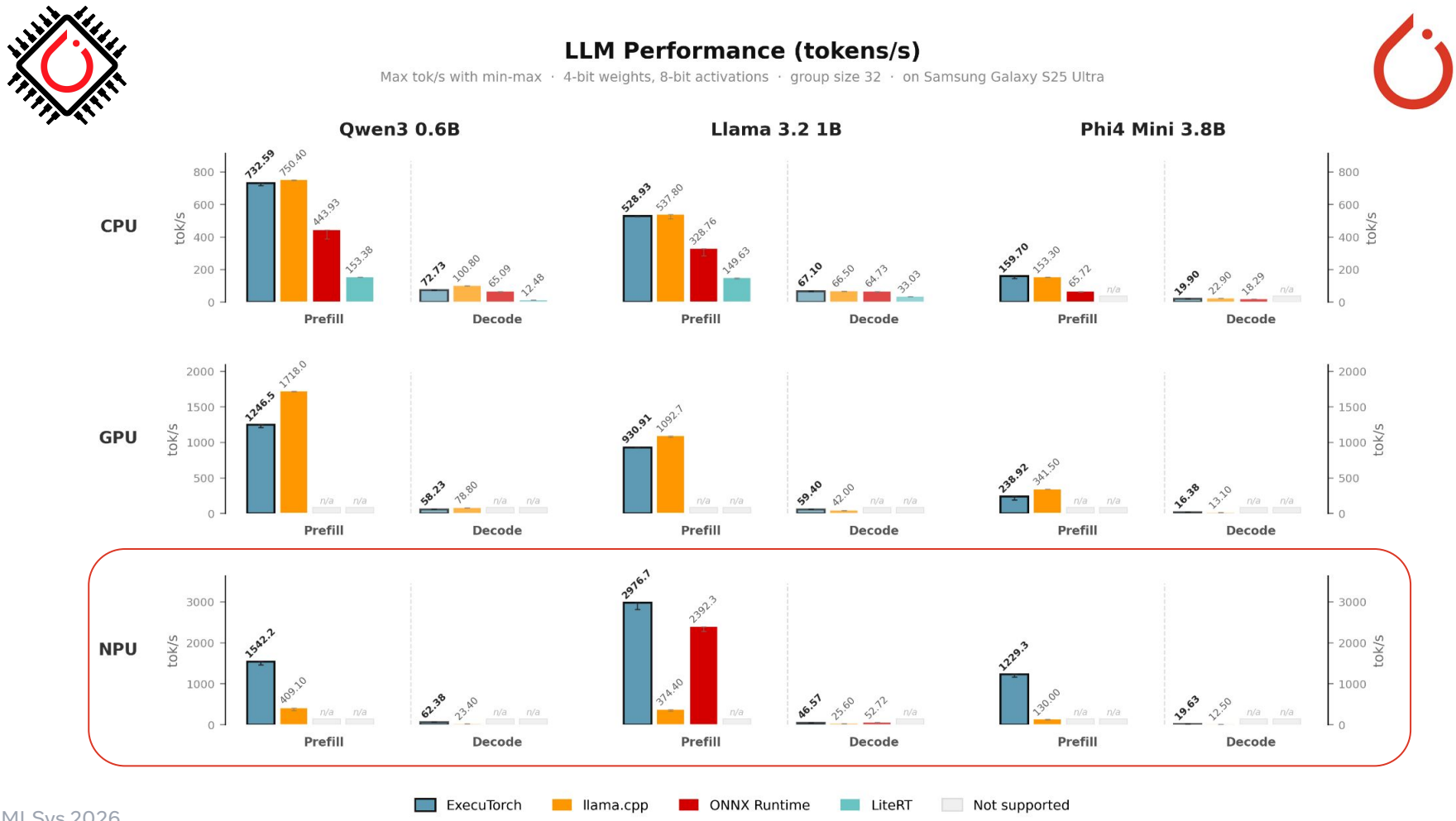
Platform	Backends	Language
Windows Linux	Vulkan, CUDA, XNNPACK Intel OpenVINO	C++
MacOS iOS	CoreML, MPS, XNNPACK	C++, Swift, Objective-C
Android	Vulkan, XNNPACK, Arm VGF, Qualcomm NPU, MediaTek NPU, Samsung Exynos	C++, Java, Kotlin
Embedded Systems	Cortex-M, Arm Ethos-U, NXP NPU, Cadence DSP	C++

Component	FP32 Portable	INT8 CMSIS-NN
Model (.pte)	103.7	29.1
ET Runtime	25.7	13.1
Kernel Registration	0.3	2.9
CMSIS-NN Library	—	35.8
Cortex-M Ops	—	5.9
System (Pico SDK + libc)	123.3	116.0
Total Flash	253	203

LLM Performance (tokens/s)

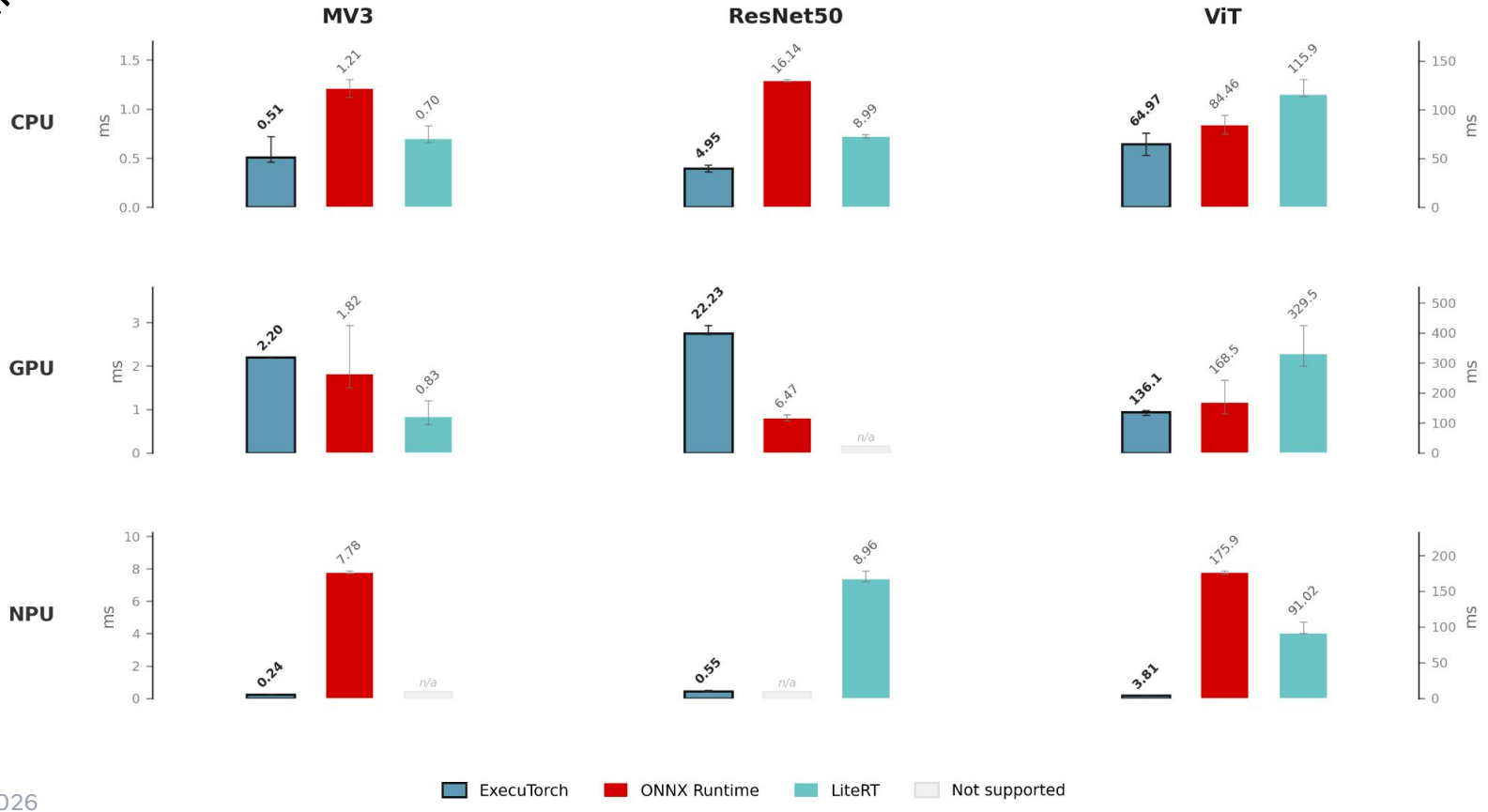
Max tok/s with min-max · 4-bit weights, 8-bit activations · group size 32 · on Samsung Galaxy S25 Ultra




LLM Performance (tokens/s)
Max tok/s with min-max · 4-bit weights, 8-bit activations · group size 32 · on Samsung Galaxy S25 Ultra
Qwen3 0.6B
Llama 3.2 1B
Phi4 Mini 3.8B
CPU
GPU
NPU
tok/s
Prefill
Decode
732.59
750.40
443.93
153.38
72.73
100.80
65.09
12.48
528.93
537.80
328.76
149.63
67.10
66.50
64.73
33.03
159.70
153.30
65.72
n/a
19.90
22.90
18.29
n/a
1246.5
1718.0
58.23
78.80
930.91
1092.7
59.40
42.00
238.92
341.50
16.38
13.10
1542.2
409.10
62.38
23.40
2976.7
374.40
2392.3
46.57
25.60
52.72
1229.3
130.00
19.63
12.50
ExecuTorch
llama.cpp
ONNX Runtime
LiteRT
Not supported

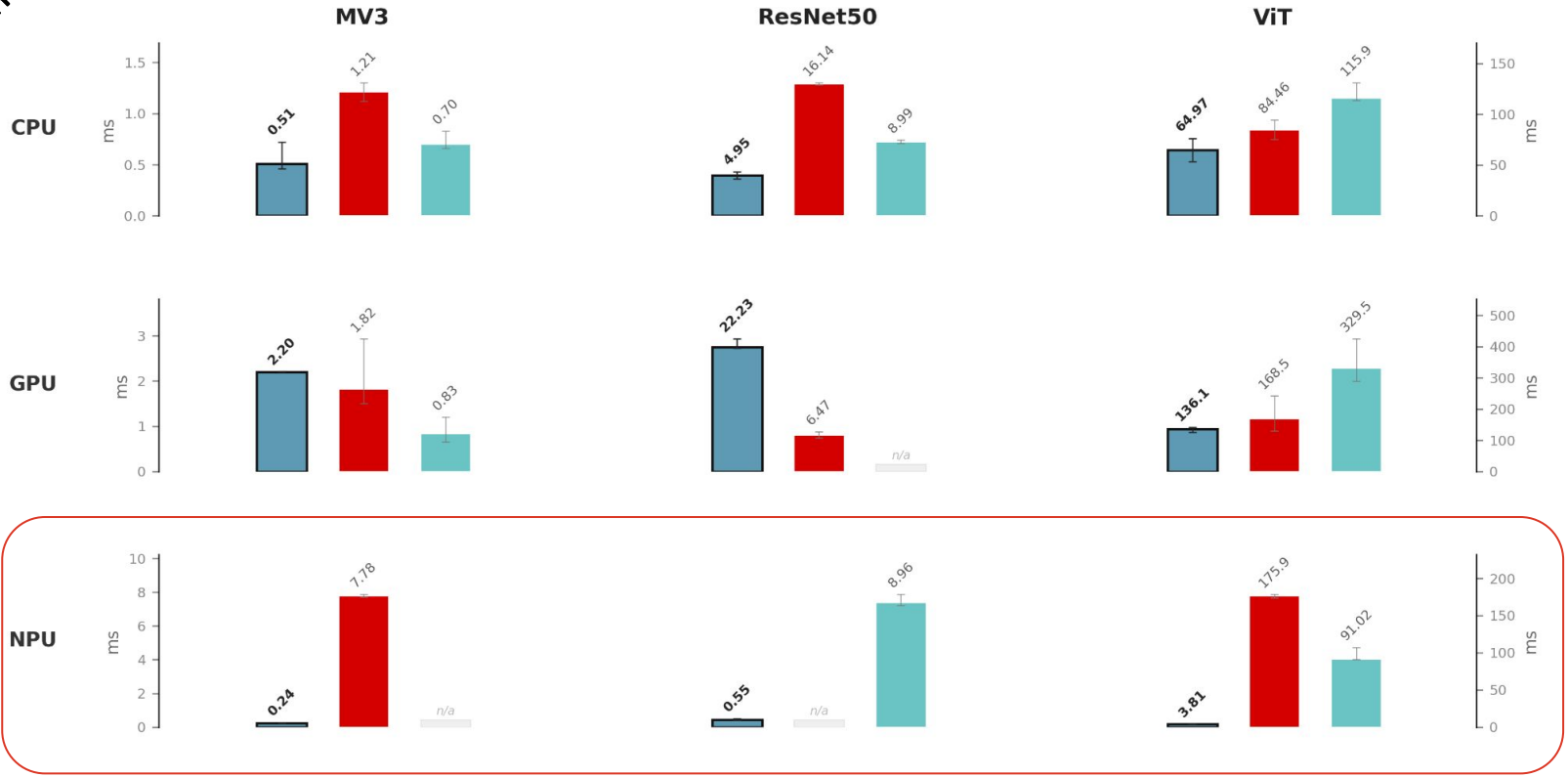
Inference Latency (ms)

Avg with p5-p95 · CPU int8, GPU fp16, NPU int8 · lower is better · on Samsung Galaxy S25 Ultra



Inference Latency (ms)

Avg with p5-p95 · CPU int8, GPU fp16, NPU int8 · lower is better · on Samsung Galaxy S25 Ultra



Current Status

- Last release v1.2 (Apr 1st) - General Availability
- We have been enabling latest LLM, multimodal models
  - Voxtral Realtime, Gemma 4, Qwen 3.6
- Successful deployments
  - Meta (Apps, Smart Glasses, Quest), LiquidAI, Private Mind, Nimble Edge
- Ecosystem integrations
  - Hugging Face, Unsloth, TorchAO
- Now part of the PyTorch foundation, embracing openness
- Just like PyTorch, accelerating research to production
- Productive hardware partnerships

Meta

Open Source

ExecuTorch Adoption in Reality Labs: Powering On-Device AI Across Meta Devices

November 21, 2025 · 10 minute read

At Meta, on-device AI is a cornerstone for delivering fast, private, and intelligent experiences to the people who use our products. ExecuTorch, our open source, lightweight, and efficient inference engine, has been instrumental in enabling these capabilities across our various apps. Today, we're excited to share that ExecuTorch is now enabling cutting-edge machine learning experiences across Reality Labs' portfolio of VR headsets and AI glasses. ExecuTorch lets developers easily deploy state-of-the-art machine learning models, unlocking richer, smarter, and more immersive experiences. By streamlining the path from

arm Developer Develop Blogs Community CPU & Hardware Support Documentation

COMMUNITY Forums Blogs Groups Help

Home / Community / AI Blog

October 22, 2025

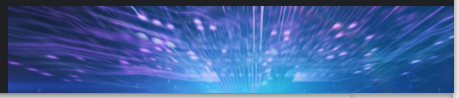
ExecuTorch 1.0 is here and with SME2 optimizations through KleidiAI

Today marks an exciting milestone with the official general availability (GA) release of ExecuTorch 1.0, a lightweight, production-ready runtime from the PyTorch ecosystem.

By Gian Marco Iodice

KleidiAI SVE

Share



Liquid

SOLUTIONS + TECHNOLOGY + CASE STUDIES COMPANY +

PRODUCT

How Liquid AI Uses ExecuTorch to Power Efficient, Flexible On-Device Intelligence

AUTHORS

Liquid AI

PUBLISHED

October 22, 2025

The Challenge: Finding an Inference Engine for Hybrid Architectures

The Solution: ExecuTorch's Enterprise-Grade Automation

The Results: Measurable Performance Gains

The Challenge: Finding an Inference Engine for Hybrid Architectures

When in early 2025 we started developing LFM2, our first generation of open-weight models for edge AI, we faced a critical challenge: existing inference engines couldn't handle the diverse architectures we needed to profile during neural architecture search. Our goal was to create the first generative AI

October 22, 2025

Arm neural technology in ExecuTorch 1.0

With the announcement of Arm neural technology, Arm is enabling neural networks and a new class of neural graphics capabilities.

By Robert Elliott

Graphics and Gaming

Share

Reading time 4 minutes

intel.

Developers / Tools / OpenVINO™ Toolkit / Optimiz

Optimizing ExecuTorch on Intel-Powered AI PCs with OpenVINO™

Authors:
Yamini Nimmagadda, Daniil Lyakhov, Mustafa Cavus, Surya Siddharth Pemmaraju, Aamir Nazir

cadence

BLOGS > SOC AND IP > RUNNING OPTIMIZED PYTORCH MODELS ON CADENCE DSPS WITH E...

Running Optimized PyTorch Models on Cadence DSPs with ExecuTorch

22 Oct 2025 · 2 minute read

By Vijay Pawar of Cadence and Matthias Cremon of Meta

POSTED ON JULY 28, 2025 TO ANDROID, IOS, OPEN SOURCE

Accelerating on-device ML on Meta's family of apps with ExecuTorch

By PyTorch Edge Team in collaboration with Family of Apps

- ExecuTorch is the PyTorch inference framework for edge devices developed by Meta with support from industry leaders like Arm, Apple, and Qualcomm.
- Running machine learning (ML) models on-device is increasingly important for Meta's family of apps (FoA). These on-device models improve latency, maintain user

arm Developer

COMMUNITY

Home / Community / AI Blog

October 22, 2025

Ethos-U and Beyond: How ExecuTorch 1.0 powers AI at the edge

AI meets the edge: ExecuTorch 1.0 brings PyTorch performance and portability to Arm's tiniest, most efficient devices.

By Per Åstrand

Feedback

SAMSUNG

Samsung Semiconductor
520,346 followers
1mo

When bringing advanced AI models to on-device environments, developers often face complex conversions and reimplementations, a time-consuming process that can slow innovation.

To overcome this challenge, #SamsungSemiconductor has been working closely with #Meta to bring #ExecuTorch — an open-source #ondeviceAI framework based on PyTorch — to Exynos mobile processors. This collaboration ensures that advanced models, from vision to large language models, can run efficiently across a wide range of edge devices.

By expanding accessibility for AI developers, this partnership empowers them to test and refine their models directly on #Exynos, unlocking new frontiers of on-device intelligence.

d devices, especially audio DSPs,
these. Cadence and Meta have

Qualcomm Products Applications Developer Support Company

Developer

Back to All

DEVELOPER BLOG

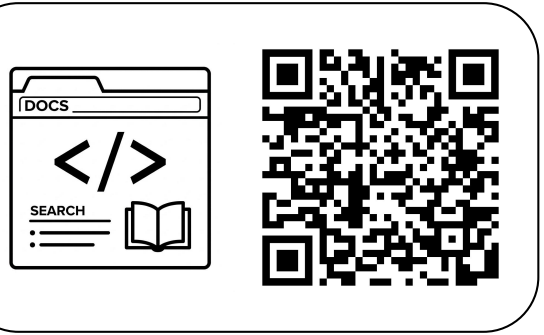
Bringing Edge AI performance to PyTorch developers with ExecuTorch 1.0

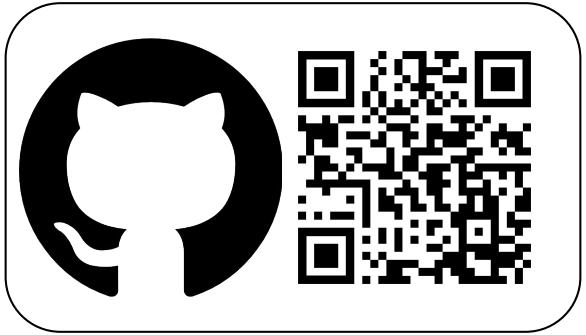
Written by Felix Baum

Written by Charlotte Mallo

Oct 22, 2025

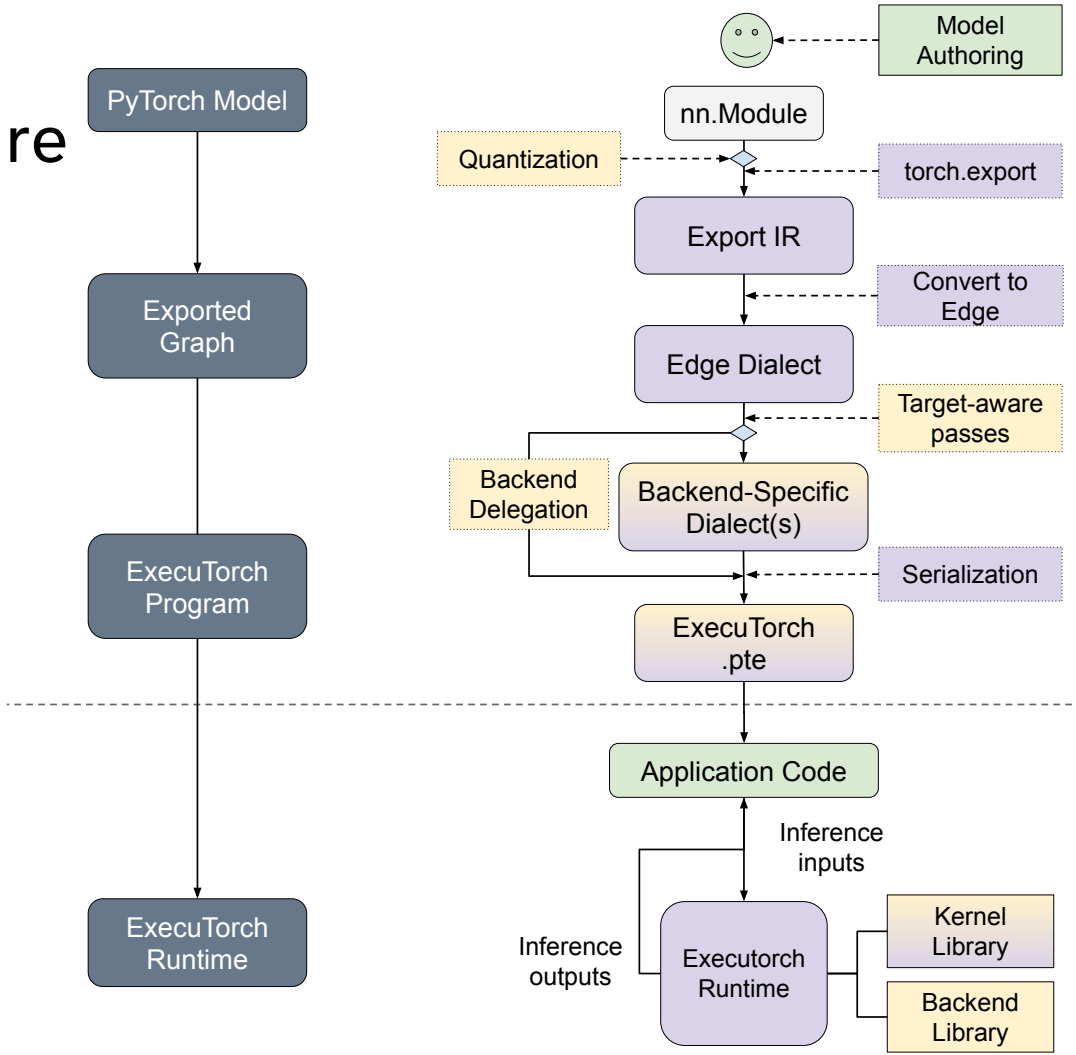
Thank you!





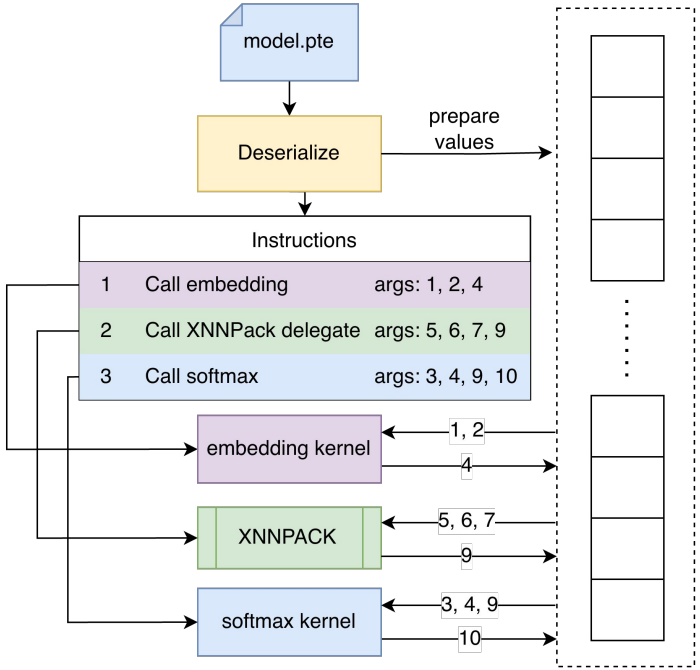
Appendix

Architecture



Model Execution

- Core Runtime
  - Light, C++17 based, portable execution engine
  - Platform Abstraction Layer for system ops
  - Aims to reduce framework tax
- Built-in Kernels
  - Portable Kernel Library
  - Op [+dtype] selective build to reduce binary size
- Language bindings
  - Python, Android (Java/Kotlin), iOS (Obj-C, Swift)
- Extensions
  - Data loaders, Module, LLM



Phase	Component	MI	ET	Speedup
Loading	Deserialization	510	97	5.3×
Loading	Initialization	312,631	8,350	37.4×
Execution	Framework overhead	324,399	75	4,325×
Execution	`aten::mul`	7,976	360	22.0×
Execution	`aten::add`	8,493	390	21.8×
Total		654,009	9,272	70.5×

Enabling Use Cases: LLM and Multimodality

- Supports models from Hugging Face via Optimum-ExecuTorch
  - Or bring your own and torch.export it
- Enabled popular models already on multiple backends
  - Llama 3.2, Qwen3, Phi 4 Mini, and many more
  - Voxtral, Gemma 3
- Key features
  - Flash Attention, Sliding window Attention
  - Quantized KV-cache
  - Speculative Decoding
  - Dynamic Shape support (or separate methods for prefill/decode)

Enabling Use Cases: Vision and Embedded

- Vision
  - Arbitrary PyTorch Model support
  - Classical Vision models run out of the box on multiple backends
    - Mobilenet-v2/v3, ResNet, ViT, and many more
    - Supports standard quantization
- Embedded
  - Allows binary size optimizations
    - Kernel library selective build
    - Ahead-of-time memory planning
  - Performant yet tiny runtime
    - Performance optimized operators (CMSIS-NN)
    - Portable Runtime size ~20 KiB
  - Supports standard quantization

Limitations and Future work

- torch.export
  - Data Dependent Control flow
  - Dynamic Shapes
  - Custom operators
- Hardware retargetability and PTE portability
  - Ahead-of-time compilation especially for NPUs
- Supporting more capable devices e.g. Laptop/Desktops
  - CUDA and Metal
  - Leveraging AOTI
- Sparsity