



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, Siddhartha 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





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

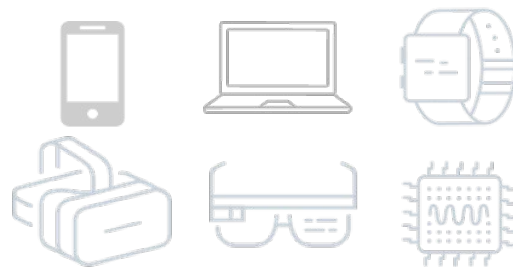
Well-defined workflow for program capture, transform, and execution with OOTB components.



Existing Solutions

PyTorch
Program

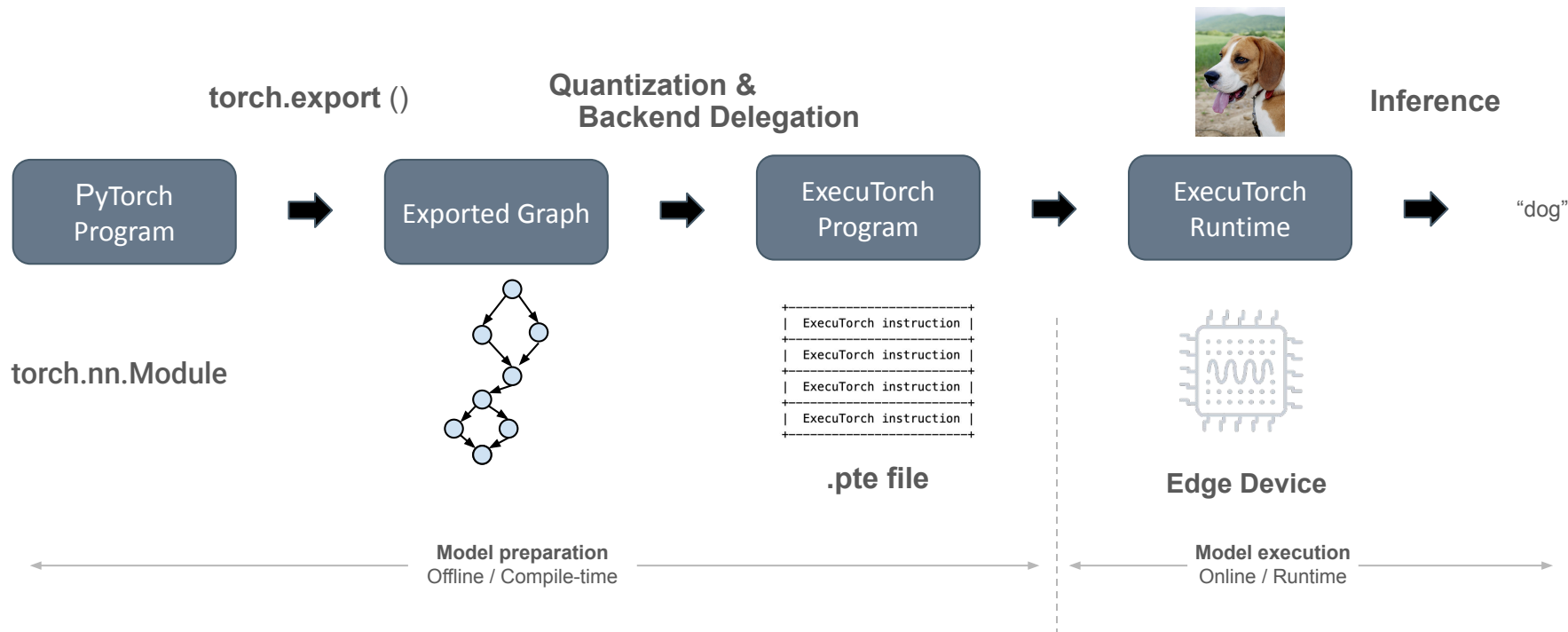
`torch.nn.Module`



Conversion-based
Compiler-based
Vendor-specific
Model-Specific



ExecuTorch





Key Architectural Choices

- Building ExecuTorch on top of PyTorch 2.0 and `torch.export`
- Introducing Graph based, Backend-aware Quantization
- Ahead-of-Time, Composable 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



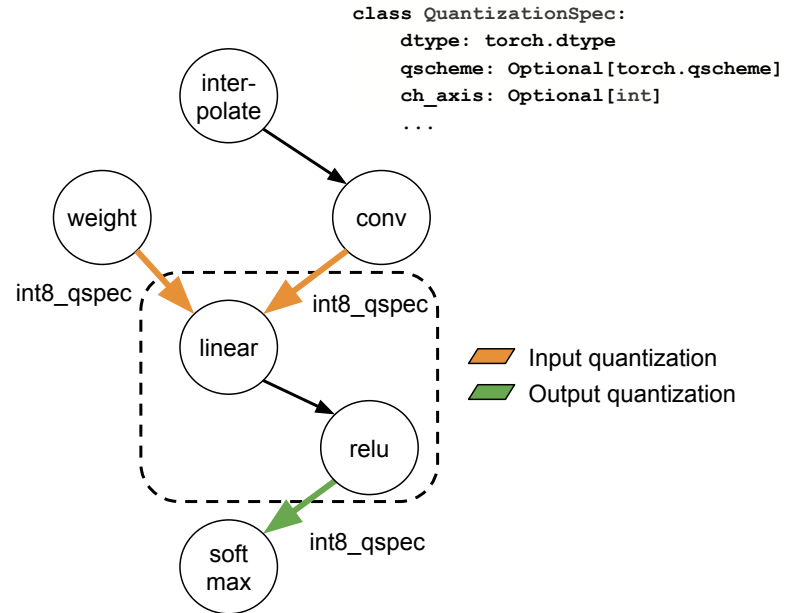
Graph Quantization

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.



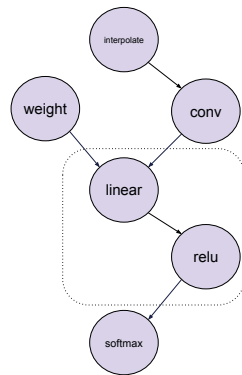
Backend Delegates

What does this design choice enable?

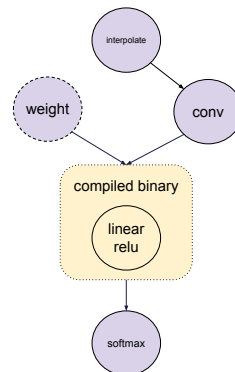
- Abstracts away hardware heterogeneity, uses same IR
- Enables vendor tooling
- Simpler core runtime, allowing it to be portable

How?

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

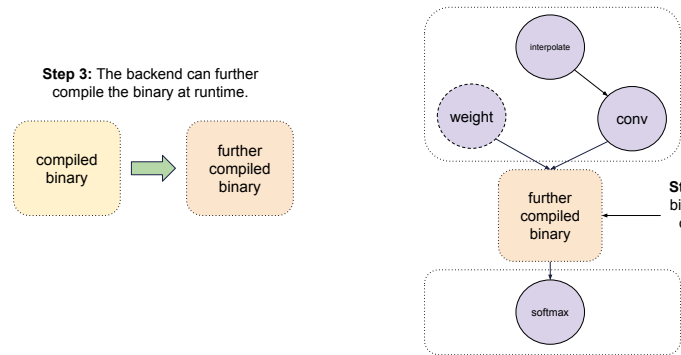


Step 1:
The backend tags
the supported
operators in the
graph.



Step 2:
The backend runs
passes to fuse parts
of the subgraph. It
will then compile
the graph into a
binary and be
embedded as part
of the PTE file.

Model preparation
Offline / Compile-time



Step 3: The backend can further
compile the binary at runtime.

Model execution
Online / Runtime

Step 4: Delegated
binary is executed
on the backend
hardware.



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

google/XNNPACK



Qualcomm



cadence



OpenVINO

NPU

- **Qualcomm QNN:** Hexagon HTP
- **Arm Ethos-U:** Ethos-U55/U85
- **Mediatek Neupilot:** 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 (PAL) for system ops
- Designed 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



Code snippets - Ahead of Time

```
import torch

# Your existing PyTorch model
model = MyModel().eval()
example_inputs = (torch.randn(1, 3, 224, 224),)

# Export to create semantically equivalent graph
exported_program = torch.export.export(model, example_inputs)
```

```
from executorch.exir import to_edge_transform_and_lower
from executorch.backends.xnnpack.partition.xnnpack_partitioner import XnnpackPartitioner

program = to_edge_transform_and_lower(
    exported_program,
    partitioner=[XnnpackPartitioner()]
).to_executorch()

# Save to .pte file
with open("model.pte", "wb") as f:
    f.write(program.buffer)
```

- Easily swap with different backends
- Insert backend specific quantization
- Power users:
 - Compiler passes
 - Memory planning





Code snippets - Runtime

```
#include <executorch/extension/module/module.h>
#include <executorch/extension/tensor/tensor.h>

Module module("model.pt");
auto tensor = make_tensor_ptr({2, 2}, {1.0f, 2.0f, 3.0f, 4.0f});
auto outputs = module.forward(tensor);
```

```
import ExecuTorch

let module = Module(filePath: "model.pt")
let input = Tensor<Float>([1.0, 2.0, 3.0, 4.0], shape: [2, 2])
let outputs = try module.forward(input)
```

```
val module = Module.load("model.pt")
val inputTensor = Tensor.fromBlob(floatArrayOf(1.0f, 2.0f, 3.0f, 4.0f), longArrayOf(2, 2))
val outputs = module.forward(EValue.from(inputTensor))
```

Customizations

C++

Compile with different backends, without changing user code

Selective build

Swift

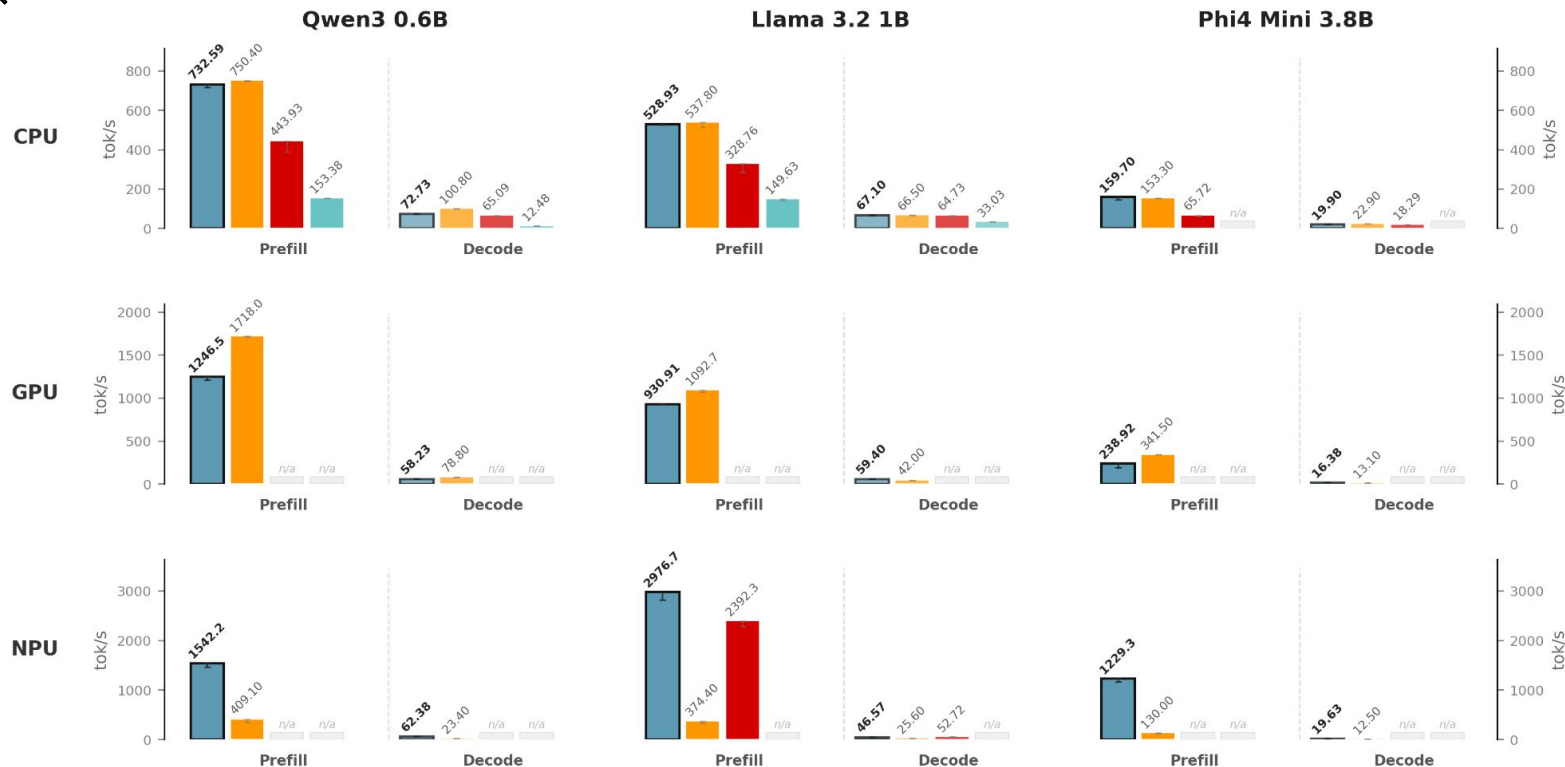
Custom operator linking

Kotlin



LLM Performance (tokens/s)

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

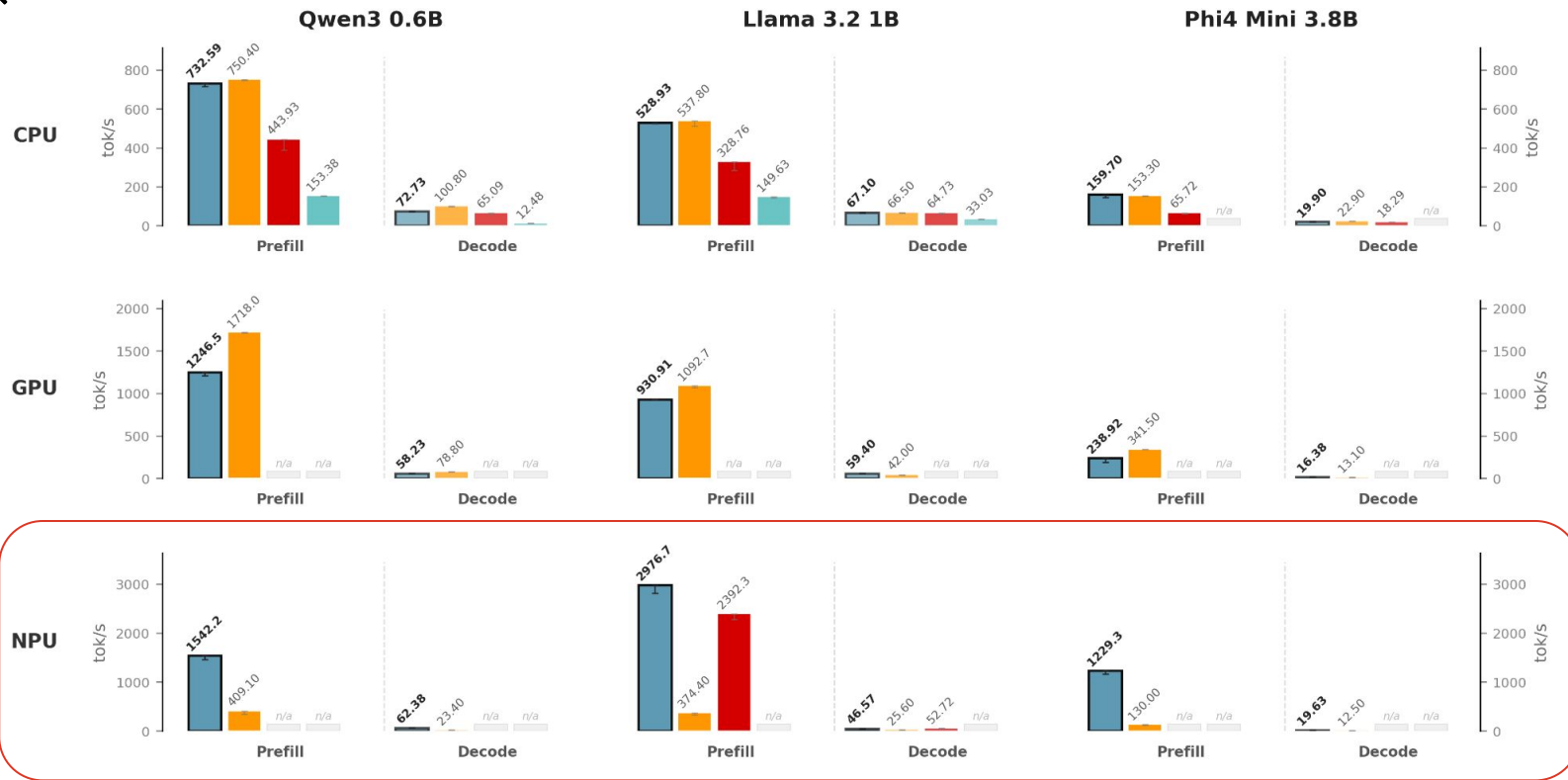


ExecuTorch llama.cpp ONNX Runtime LiteRT Not supported



LLM Performance (tokens/s)

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

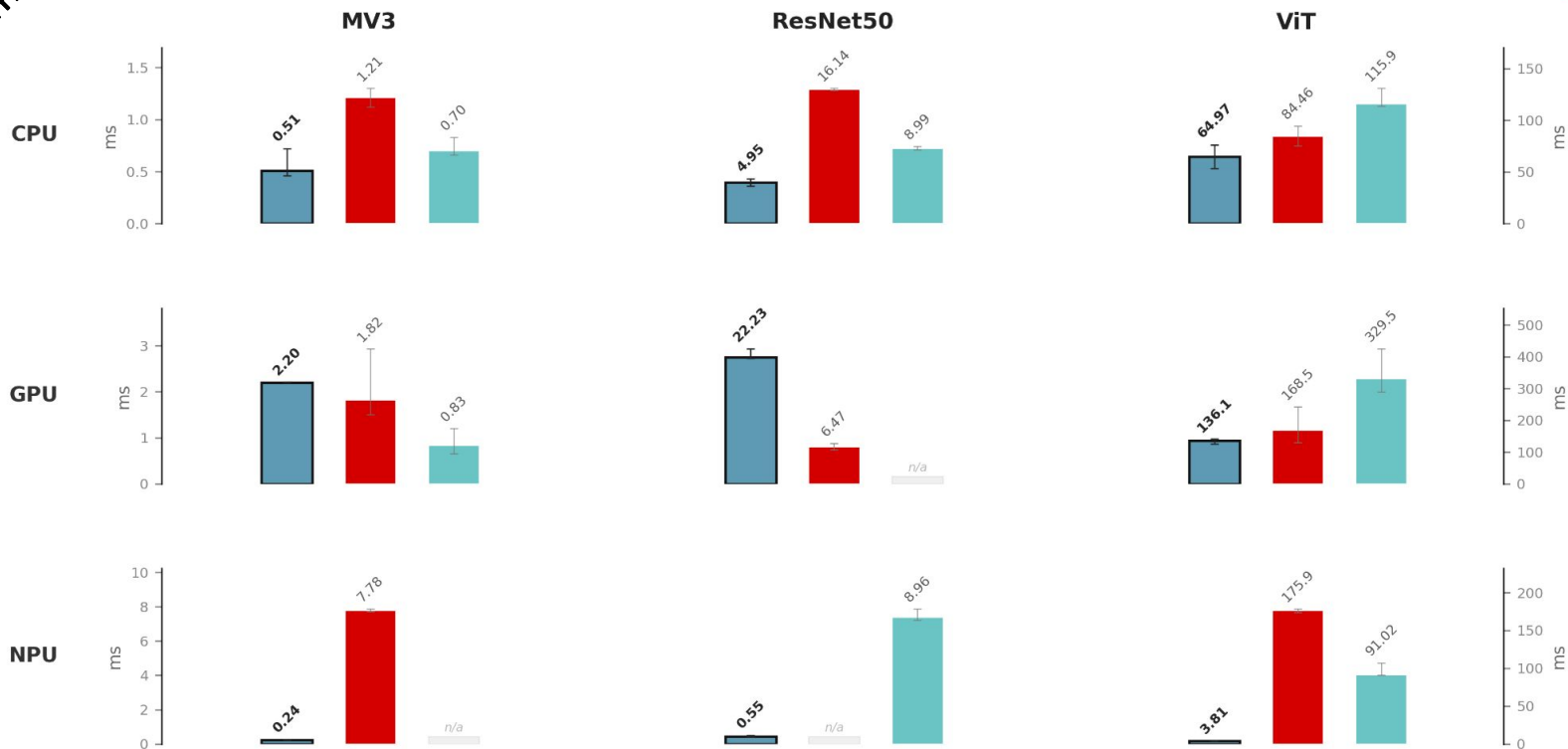


ExecuTorch llama.cpp ONNX Runtime LiteRT Not supported



Inference Latency (ms)

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

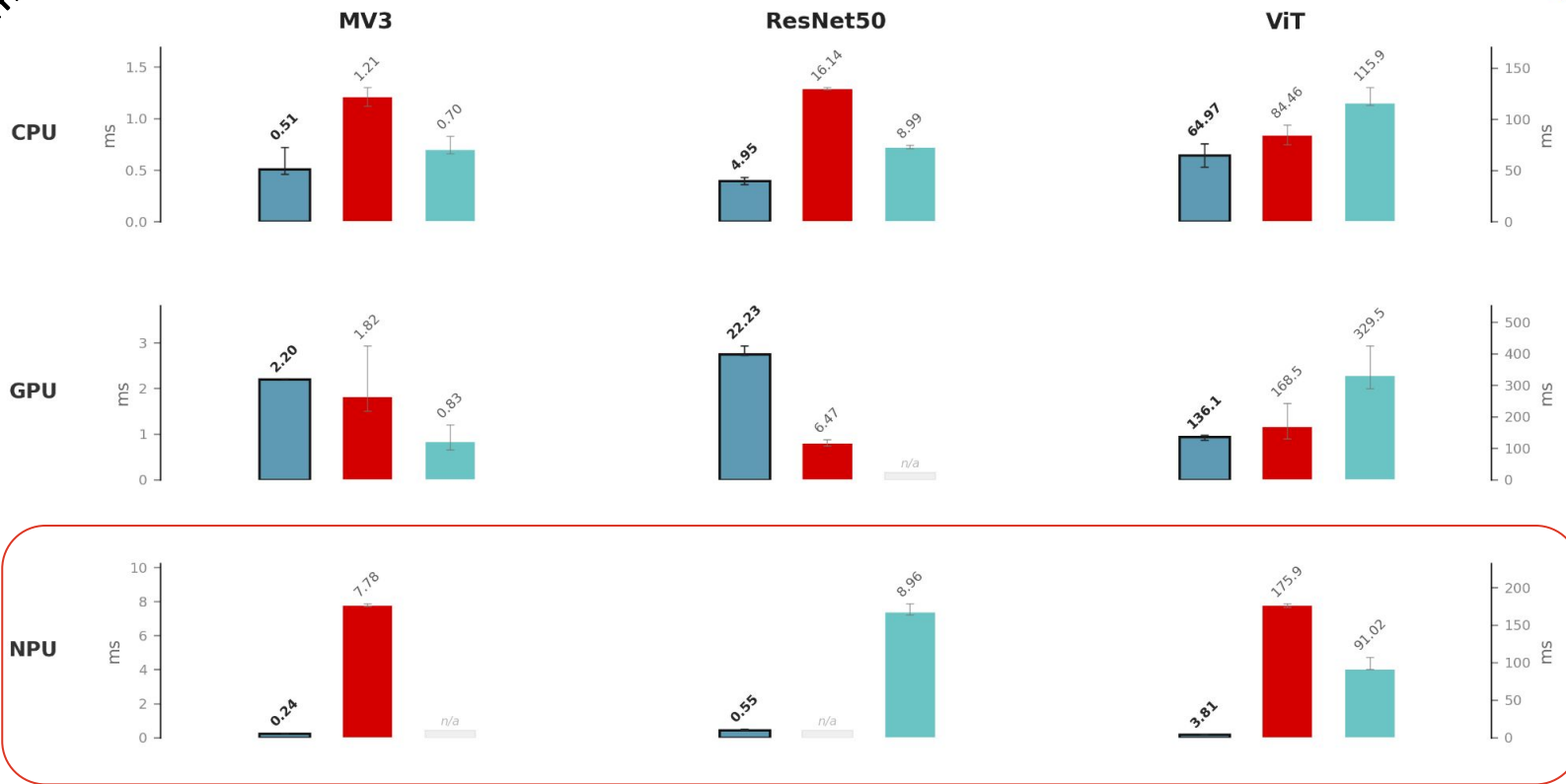


ExecuTorch ONNX Runtime LiteRT Not supported



Inference Latency (ms)

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





Current Status

- Last release v1.3 (May 21st) - General Availability
- We have been enabling latest LLMs, 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



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 research to applications and allowing people to get our work in their hands, ExecuTorch is

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



arm Developer Community CPU & Hardware Support Documentation

COMMUNITY Forum Blog 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 [ianhasek](#)

Share X in f e p

intel

Developers Tools OpenVINO™ Toolkit Optim

Optimizing ExecuTorch on Intel-Powered AI PCs with OpenVINO™

Print Email

Authors:

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

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.

SOLUTIONS TECHNOLOGY CASE STUDIES COMPANY

PRODUCT

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

LIQUID AI

October 22, 2025

The Challenge: Finding an Inference Engine for Hybrid Architectures

The Solution: ExecuTorch's Enterprise-Grade Substrate

The Results: Measurable Performance Gains

The Challenge: Finding an Inference Engine for Hybrid Architectures

When in early 2025 we started developing LPM2, our first generation of open-weight models, AI, we faced a critical challenge: existing inference engines couldn't handle the diverse architectures needed to profile during neural architecture search. Our goal was to create the first generation of efficient LLM-based models that could run on-device.

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

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 X in f e p

Reading time 4 minutes

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

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 Oct 22, 2025

Written by Charlotte Mallo



Thank you!

