ExecuTorch 1.0 - General Availability (GA)

On-device AI inference for PyTorch

Mergen Nachin - Software Engineer, PyTorch team at Meta
Bilgin Cagatay - Engineering Manager, PyTorch team at Meta

Why On-Device AI?

1. Enhanced Privacy

Examples

- Health tracking
- Legal document analysis
- Meeting transcription

2. Remote areas and accessibility

Examples

-Emergency Response during natural disaster
-Portable medical device with real-time image analysis
-Educational and accessibility tools

3. Real-time/low-latency

Examples

- Wearable devices detecting falls for elderly
- Noise cancellation headphones
- Live Translation

4. Cost

Examples

- Save on server cost
- Abundance of compute availability on the edge

5. Model are smaller and smarter

Examples

-Llama 3.3 70B vs Llama 3.1 405B

On-Device AI fundamental challenges

1.Battery-life, Power

Examples

-Tiny batteries, e.g., 100-500mAh
-Need to charge or swap batter often

2.Memory

Examples

-Large models but small memory, e.g., ~4GB RAM
-Memory bandwidth limitations

3.Thermal Constraints

Examples

-Device becomes too hot quickly -> safety
-Throttling affects speed and accuracy

4. Hardware Heterogeneity

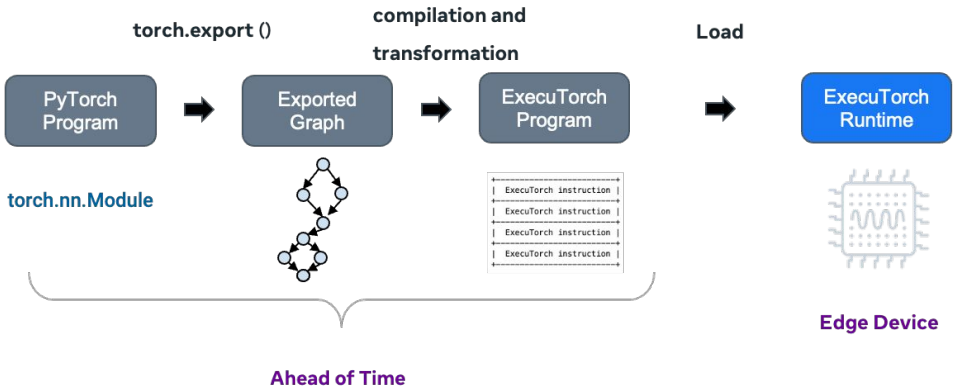
Examples

- Different chipsets (e.g., Qualcomm, Apple, MediaTek, Arm, Samsung, Intel, STM, NXP)
-General purpose programmable CPU and GPU vs special purpose NPU
-Need to optimize for each platform separately

ExecuTorch

How to accelerate Research-To-Production process?

No conversions or rewriting



Code snippets - Ahead of Time

```python
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)
```

```python
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)
```

Customizations

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

Code snippets - Runtime

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

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

C++

```swift
import ExecuTorch

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

Swift

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

Kotlin

Customizations

Compile with different backends, without changing user code

Selective build

Custom operator linking

Design Principles

- **Productivity**
  - Use familiar PyTorch toolchains for a seamless E2E experience from authoring → deployment
- **Portability**
  - Support a wide variety of edge platforms, from powerful mobile chipsets to embedded systems
- **Performance**
  - Leverage hardware acceleration via a lightweight runtime
- **Open Source and Community Driven**
  - Develop in the open, encouraging community contributions and feedback
- **Modularity**
  - Provide well defined workflow for PyTorch programs to be captured, expressed, transformed, and executed, with out-of-the-box components

Timeline

Preview / MVP - Oct 2023
Hardware partnerships
Early user feedback
Code access

Alpha - Feb 2024
Delegate enhancements
Rich developer SDK
Llama support

Beta - Oct 2024
API stability
Strong performance
Early adoption

General Availability
Oct 2025
Backend coverage and reliability
Community and ecosystem
Adoption

ExecuTorch 1.0 - General Availability

Success Stories

Adoption




Oculus VR


Ray-Ban Meta Smart Glasses


Meta Ray-Ban Display

Liquid

Private Mind

NimbleEdge

Ecosystem Integration

React Native ExecuTorch

Hugging Face transformers and optimum

torchao

Unsloth AI

Ultralytics

NVidia Flare

Digica

12 Backends Total

From Beta -> Production-ready (reliability and coverage)

XNNPACK with Arm Kleidi for CPU acceleration
Apple Core ML for Apple silicon
Qualcomm AI Engine for Qualcomm Hexagon NPU
Arm Ethos-U NPU
Vulkan GPU

Alpha or Beta status

Arm VGF
NXP Semiconductors' eIQ Neutron NPU
Samsung Exynos NPU
OpenVINO from Intel

Newly added

Cadence DSP
MediaTek NPU
Apple Metal Performance Shaders (MPS)

ExecuTorch 1.0 - General Availability

Featured improvements since Beta

Reliability and hardening

Strong coverage and performance on Qualcomm AI Engine backend on LLMs

Conformer (i.e., transformer based audio transcription) run efficiently on low -power Arm ethos-U backend

Arm Kleidi SME2 enabled

Windows support

UX - pip/SwiftPM/Maven

Program-data separation (e.g., LoRa)

torchao interoperability (e.g., HQQ algorithm)

Javascript/Browser support via wasm (experimental)

ExecuTorch 1.0 - General Availability

📝 Text LLMs:

- Llama 3.2
- Qwen 3
- Phi-4-mini
- LiquidAI LFM2

🤗Hugging Face Optimum Integration:

- Codegen
- Gemma, Gemma2, Gemma-3-1b (and variants)
- GLM
- GPT2, GPTJ, GPTNeoX, GPTNeoXJapanese
- Granite
- Mistral
- Qwen2
- Olmo
- Phi, Phi4
- Smollm, Smollm3
- Starcoder2

🖼️ Multimodal Examples:

- Voxtral
- Llava
- Gemma3
- Whisper

🔌 LoRA Adapter

Vision Models

- Cvt
- Deit
- Dit:
- EfficientNet:
- Focalnet
- Mobilevit
- Mobilevit2
- Pvt:
- Swin

Multimodal LLM

export

Hugging Face transformers and optimum

```
optimum-cli export executorch \
  --model "mistralai/Voxtral-Mini-3B-2507" \
  --task "multimodal-text-to-text" \
  --recipe "xnnpack"
  --qlinear 8da4w
```

Kotlin

```
LlmModule llm = new  LlmModule(“model.pte”,...)
llm.load()
llm.prefillImages(image_tokens)
llm.prefillAudio(audio_tokens)
llm.prefillPrompt(text_tokens)
llm.generate(...)
```

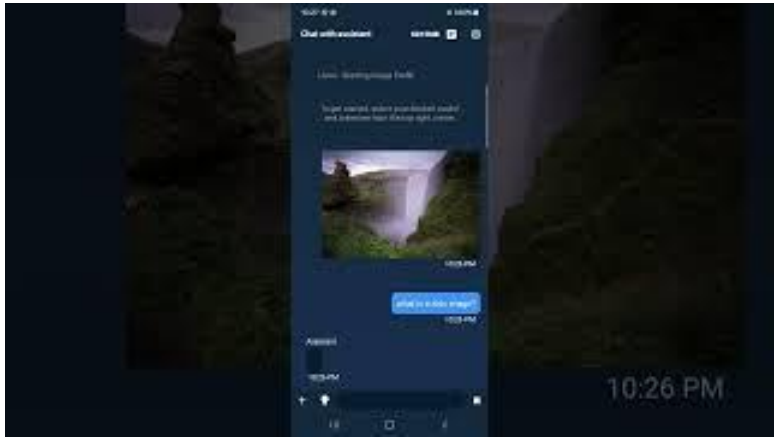
Swift

```
let runner = MultimodalRunner(“model.pte”)
try runner.load()
try runner.generate([
  MultimodalInput(image),
  MultimodalInput(Audio(...)),
  MultimodalInput(“Your text prompt”)
],...)
```

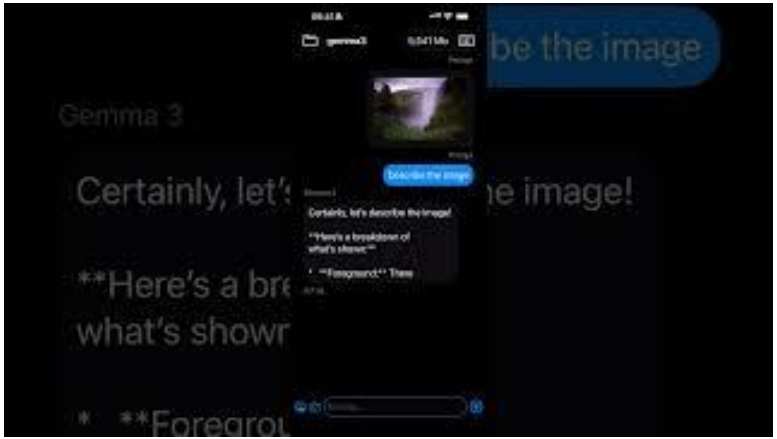
C++

```
 auto runner = create_multimodal_runner(...);
 runner->load();
 runner->generate({
  MultimodalInput(Image(std::move(imageFloats)), ...),
  MultimodalInput(Audio(std::move(audioFloats)), ...),
  MultimodalInput("Text prompt")
}, …);
```

Demo: Gemma3-4b (vision-text-input LLM)



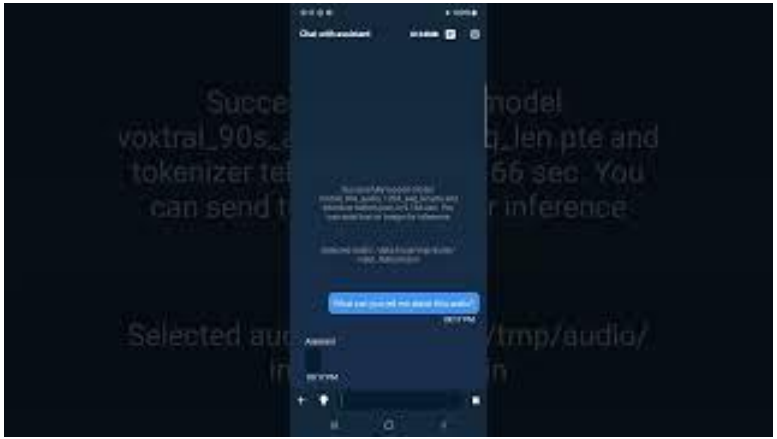
Android



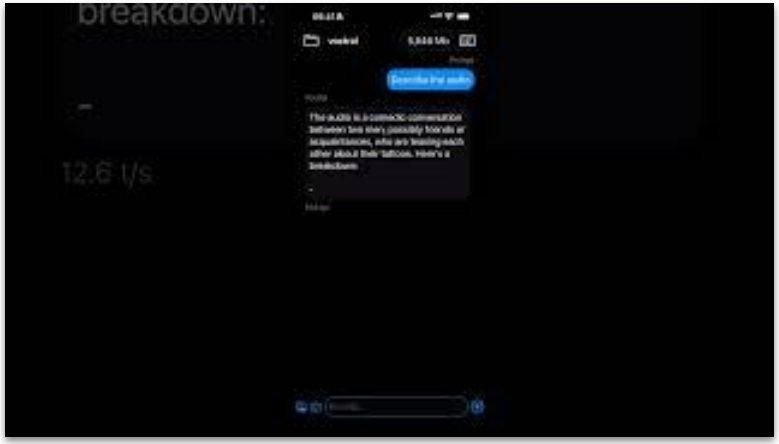
iOS

Running on XNNPACK (CPU)

Demo: Voxtral-Mini-3B-2507 (audio-text-input LLM)



Android



iOS

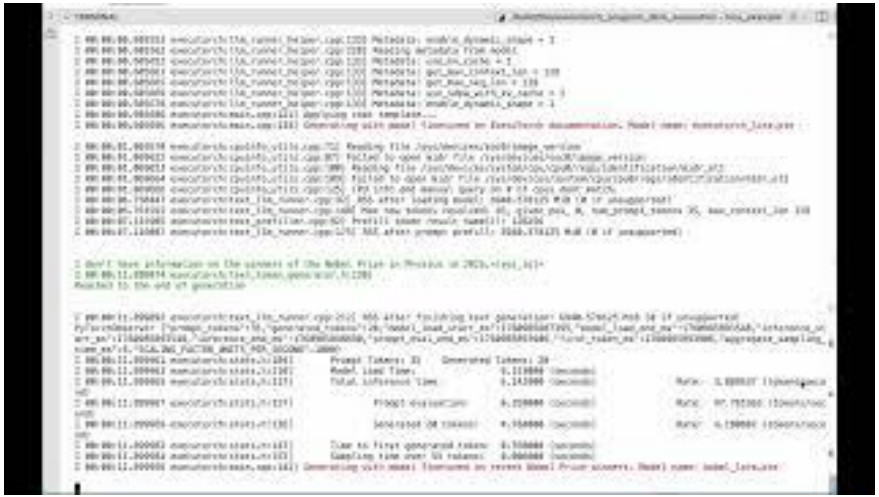
Running on XNNPACK (CPU)

LoRA capability

Demo:

- Use llama as foundation
- Two adapters fine-tuned.
  - Dataset1: On 2025 nobel prize winners
  - Dataset2: On ExecuTorch tutorials
- Can load **simultaneously**
- Save on memory and binary

[Example code]



ExecuTorch 1.1 and beyond

ExecuTorch 1.1 and beyond

- Continue bringing the remaining 8 backends to “production-ready”

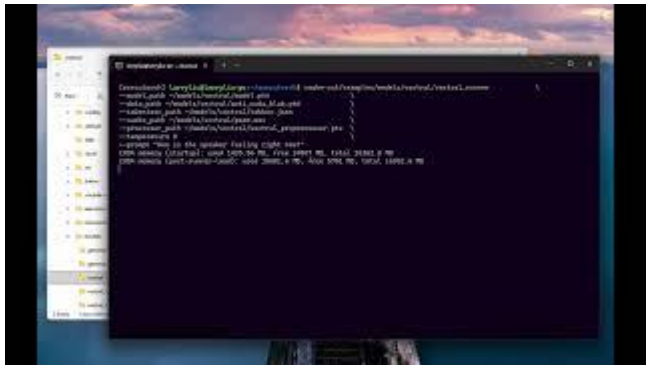
- Continue building stronger ecosystem integration
  - External partnerships with industry, academia and research
  - SDK (e.g., IoT, mobile, robotics)
  - Modeling libraries
  - AI engines and runtimes
  - Toolkits (e.g., quantization)

ExecuTorch 1.1 and beyond: Desktop/laptop

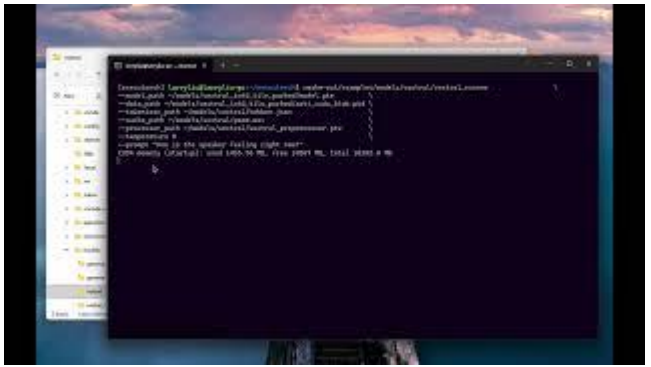
- ExecuTorch has been traditionally on mobile and embedded devices
- We’re now expanding into personal desktop/laptop (aka “AI PC”)
- Main missing backend was CUDA…. until now

Voxtral-Mini-3B-2507 (audio-text-input LLM) – on CUDA

Audio



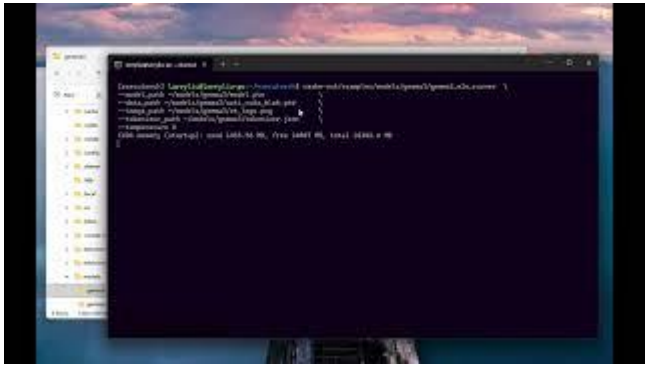
bf16



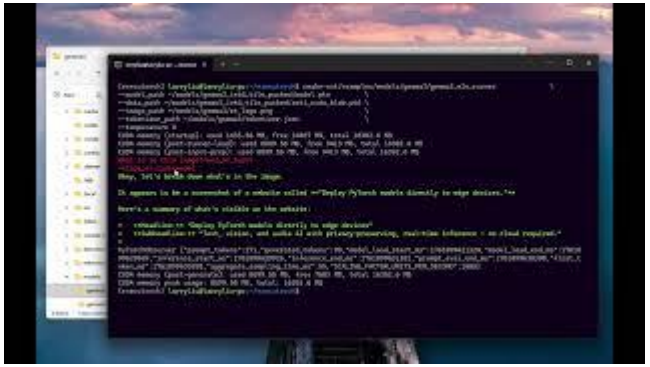
int4

Gemma3-4b (vision-text-input LLM int4) on CUDA

Image



bf16



int4

Whisper-small

Audio

bf16

int4

RTX 5080

	# of parameters	Load time	Prefill	Prefill token/s	Generate token/s	Peak GPU memory (1065 MiB baseline w/ remote desktop)	Artifact size	Binary size
Voxtral bfloat16	**~4.68B**	2496 ms	226 ms, 23s audio, 387 prompt tokens in total	1712	59	11.51 GB	9.0GB	16MB
Voxtral Int4 tile packed		2231 ms	512 ms, 23s audio. 387 prompt tokens in total	756	96	6.10GB	3.7GB	
Gemma3 bfloat16	**~4.97B**	10210 ms	640 ms, 1 image, 271 prompt tokens in total	423	62	13257 MB	9.5GB	14MB
Gemma3 Int4 tile packed		5643 ms	480 ms, 1 image, 271 prompt tokens in total	564	97	8699.56 MB	5.4GB	
Whisper-small bfloat16	**~281M**	375 ms	Encode time: 40ms, 30s audio.	N/A	145	2.83 GB	1000MB	16MB
Whisper-small int4 tile packed		371 ms	Encode time: 42ms, 30s audio.	N/A	176	2.10GB	269MB	

Desktop/laptop

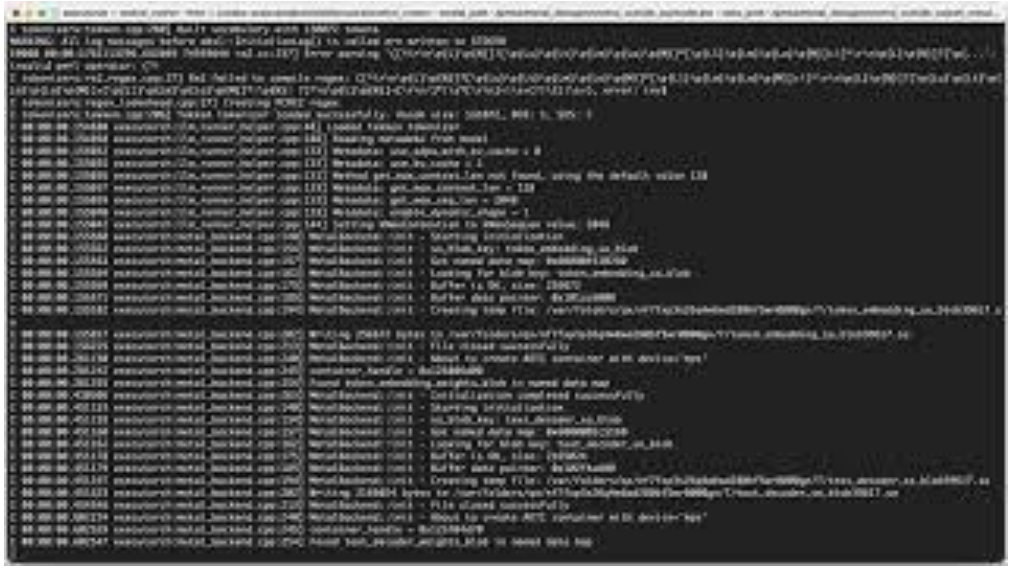
- Model agnostic. Demo on these: Voxtral, Gemma3, ResNet, Whisper
- Stay in the PyTorch ecosystem (e.g., fine-tuning, quantization, compilation)
- No python runtime during inference
- No libtorch dependency during inference, saving on binary size
- Leverages Ahead-of-Time Inductor (AOTI) to support CUDA without reinventing the wheels
- Validated on Linux and Windows (WSL). Windows native (WIP)

Desktop/laptop

- Still early but very promising direction
- We'll continue expanding model coverage and leverage the compiler technology (AoTI) from pytorch
- Currently experimenting following similar approach on Macbooks too. Leverage Metal GPU by using the AoTI

Voxtral-Mini-3B-2507 (audio-text-input LLM) – on Metal

Audio



Voxtral-Mini-3B-2507 (audio-text-input LLM) – on Metal

M4 (Metal)

	# of parameters	**Load time**	**Prefill**	**Prefill token/s**	**Generate token/s**	**Peak GPU memory**	**Artifact size**	**Binary size**
Voxtral bfloat16	**~4.68B**	2539 ms	963 ms, 23s audio, 386 prompt tokens in total	400	16	13.2 GB	9.0GB	11MB

Thank you

Integrate ExecuTorch into your product/library/framework

- pytorch.org/executorch
- github.com/pytorch/executorch

Thank you to all the partners and users!