Overarching Theme:

ExecuTorch's power comes from open source collaboration, open models, open backends, open ecosystem. The Foundation announcement is the capstone that makes this even stronger.

PyTorch CONFERENCE EUROPE 2026

Bringing ExecuTorch to the Next Frontiers of Edge AI

Mergen Nachin
Meta Superintelligence Labs

Agenda

Where we are

ExecuTorch 1.x and who's using it

Voice

Enabling the next wave of open-source voice models

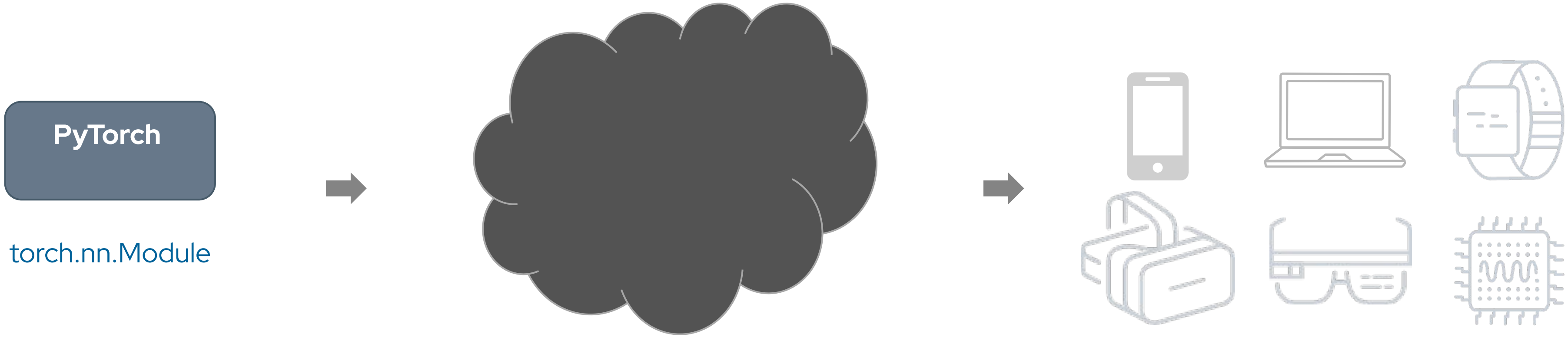
Desktop and laptop inference

Novel architectures on consumer GPUs

Ecosystem & open source

Partners, Community, and an **Announcement**

On-Device AI: Opportunity and Problems

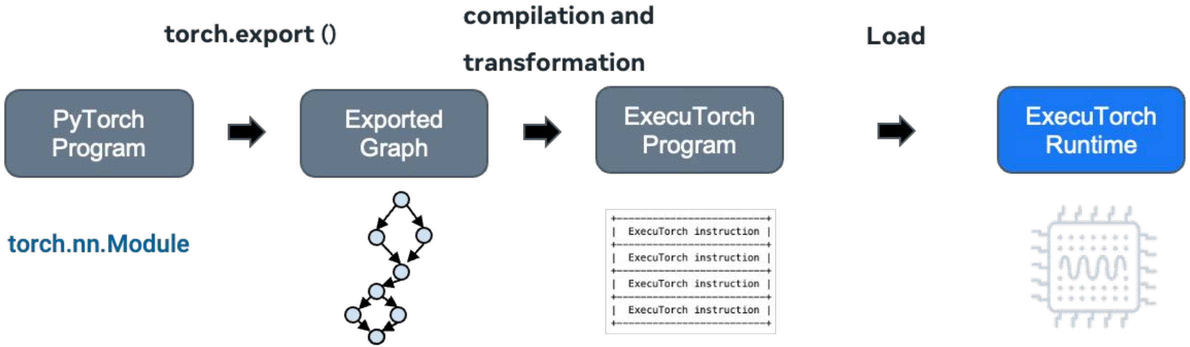


- Enhanced Privacy
- Real-time/Low latency
- Cost
- Offline/low connectivity

- Model authors rewrite inference code per platform
- Hardware vendors build siloed, proprietary SDKs
- Every new model × every new device = combinatorial explosion
- Result: significant engineering effort per deployment target

ExecuTorch 1.0 — Where We Are

Shipped General Availibility (GA) October 2025. Production-ready



No conversions or rewriting

14 hardware backends	Apple, Arm, Cadence, Intel, MediaTek, NXP, Qualcomm, Samsung, Vulkan, CUDA..
50+ reference models	LLMs, vision, multimodal, speech, generative — and growing
<50KB runtime	Bare-metal capable, low overhead
5 OS platforms	Linux, macOS, Windows, Android, iOS

Since GA, we've also significantly improved microcontroller support.

Arm Cortex-M with CMSIS-NN acceleration
Beta status now

On MobileNetV2 (Alif E8, Cortex-M55), we're outperforming TFLite Micro

On MV2, 8% faster inference, 86% faster model load, and 6% less RAM

Further expansions: quant schemes, model coverage, more perf, reducing cost/resources

Check-out multiple talks on ExecuTorch on embedded systems at this conference

[Full details]

Who is using ExecuTorch

WhatsApp

Meta Quest VR

Ray-Ban Meta Smart Glasses

Meta Ray-Ban Display

Liquid LM Studio Private Mind

Meta's Family of Apps: *Instagram (Cutouts), WhatsApp (bandwidth estimation, noise canceling), Messenger (E2EE language identification), Facebook (SceneX image understanding). Billions of users.*

Meta Reality Labs: *Quest 3/3S (depth estimation, scene understanding, hand tracking), Meta Ray-Ban Display (live translation, visual captions), Oakley Meta Vanguard (real-time performance insights).*

Liquid AI: *Deployed LFM2 models (350M–4B) on AMD Ryzen AI and Samsung S24 with 2x faster inference and lower latency.*

LM Studio: *Shipping real-time voice transcription powered by ExecuTorch in their desktop application.*

Private Mind: *100% on-device AI assistant via React Native ExecuTorch.*

Voice — A New Frontier

Open-source voice models are proliferating — and fast

Lab	Models
Meta/FAIR	Seamless, Omnilingual ASR, SAM Audio
Mistral	Voxtral Realtime, Voxtral TTS
NVIDIA	Parakeet TDT, Sortformer, Canary
OpenAI	Whisper
Cohere	Transcribe
IBM	Granite Speech
Alibaba	Qwen3-ASR, Qwen3-TTS

All open source

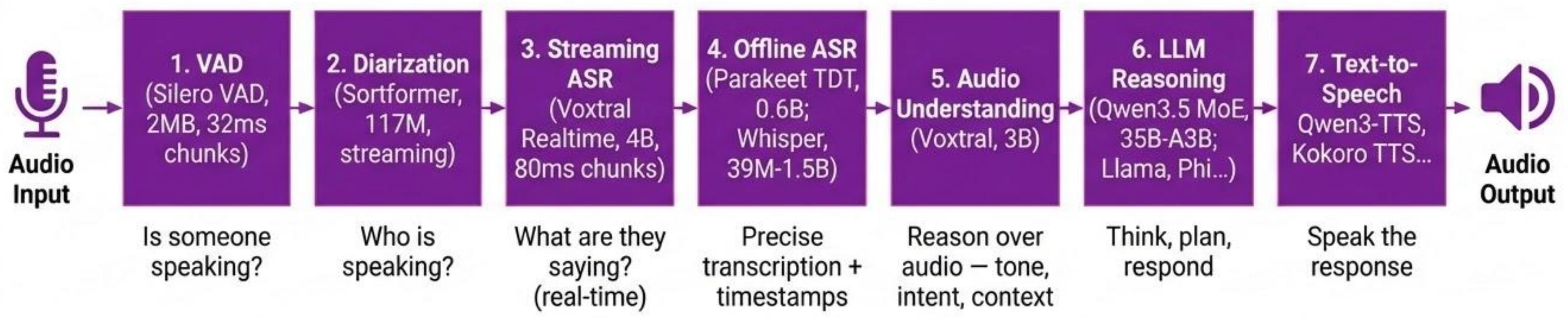
Lack of unified native deployment story.

8.4B voice assistants active worldwide Statista.

Voice AI funding surged 8x to $2.1B in 2024 CB Insights

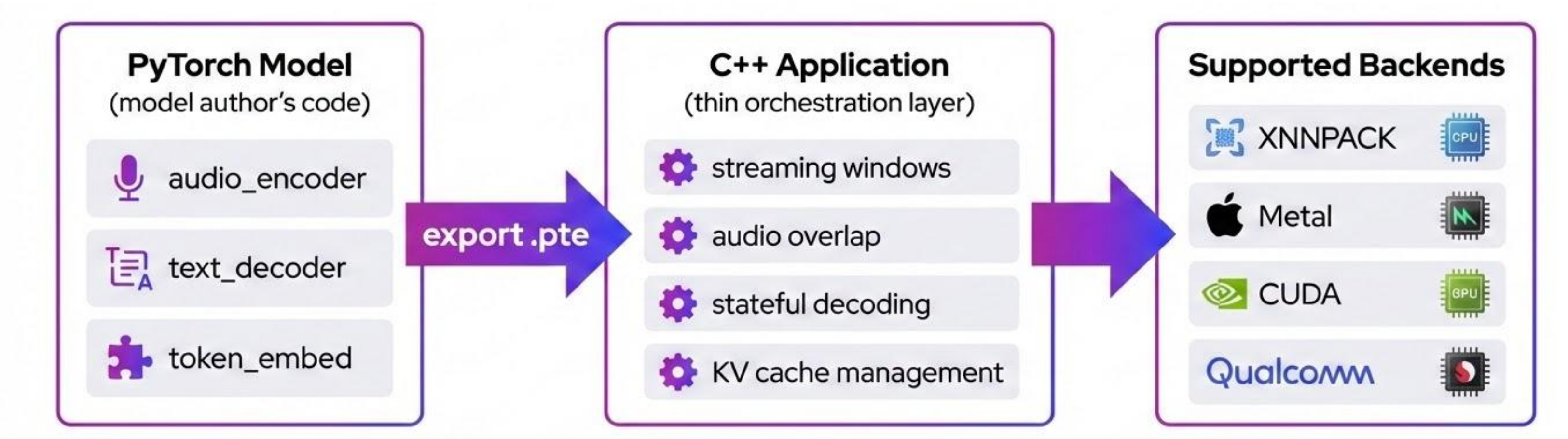
Voice Agent Building Blocks

Composable voice agent pipeline



Case Study: How We Enable Voice Models

Export the model, orchestrate in C++



Minimal edits to be torch.export() compatible

Small binary, no Python overhead

Multiple backends, same code

Demo – Realtime voice transcription

- Built on top of Voxtral-Mini-4B-Realtime-2602, int4 quant
- Same cpp code runs on CPU, Metal and CUDA
- MacOS, Windows, Linux

[DEMO]

Demo link1, link2

Cross Platform, Multi-backend

The approach we showed for Voxtral applies to every model we support.

Model	Task	Backends
Voxtral Realtime	Streaming ASR	XNNPACK, Metal, CUDA
Parakeet TDT	ASR + timestamps	XNNPACK, Metal, CUDA, Vulkan
Sortformer	Diarization	XNNPACK, CUDA
Whisper	Offline ASR	CUDA, Metal, CPU, Qualcomm HTP
Silero VAD	Voice activity	XNNPACK
Conformer ASR	Transcriptions	Arm Ethos-U

Unified cross-platform, multi-backend deployment one export path, one C++ runtime, cross platform, multi-backend.

ExecuTorch on Desktops and Laptops

*Voice showed ExecuTorch's **breadth** – many models, many backends*
*Now let's **deep dive** into inference on desktops and laptops*

The demand: local inference on consumer hardware is growing – native apps need to embed AI without cloud dependency, Python runtime, or libtorch.

Can we use the **same principles** from ExecuTorch?

- Model agnostic – any model that is compatible with torch.export
- No Python runtime – ideal for native C++ applications embedding AI
- No libtorch dependency – self-contained binary, smaller binary footprint
- PyTorch ecosystem – fine-tuning, quantization, compilation all stay within the same ecosystem
- Multi-backend – CUDA and Metal via AOTInductor
- Leverage other DSLs – Bring your own triton kernels easily

Example: Qwen3.5-35B-A3B

35 billion parameters, 3 billion active, mixture of experts

Decoder Layer

- Attention (hybrid – two mechanisms coexist)
    - GatedDeltaNet 75%
      Linear attention with recurrent state
    - Standard Attention 25%
      Softmax with KV cache

- Sparse Mixture of Experts
    - Router picks 8 of 256 experts per token
    - Shared experts

Why is this an interesting model?

- Capable model

- Small active parameters that can fit in consumer GPU

- Memory doesn't grow in relation to context length due to linear attention

- Similar techniques are appearing in other models like GLM-4.7-Flash, Nemotron-Cascade-2-30B-A3B

Q: Can we enable this model on ExecuTorch easily using same principles?

The Model Is Just PyTorch (and possibly custom kernels)

```python
# The entire written in standard PyTorch
def forward(self, tokens, input_pos):
    x = self.embed_tokens(tokens)
    for layer in self.layers:
        x = layer(x, input_pos)
    x = self.norm(x)
      return self.lm_head(x)


# GatedDeltaNet linear attention written in triton.
output, state = torch.ops.triton.chunk_gated_delta_rule(q, k, v, g, beta, state)

# Fused MoE triton kernel — loads only 8 of 256 experts
result = torch.ops.triton.fused_moe(x, w1, w1_scale, w2, w2_scale, ...)
```

They are compatible with torch.export()

Optimizations on multiple layers. Flexibility and performance

Example 1: Model-level

Quantization, pruning, distillation. Works with TorchAO or any PyTorch-native optimizations

```
# 4 bit quantization from 67 GB → 17 GB
from torchao.quantization.quant_api import quantize_
quantize_(model, Int4WeightOnlyConfig(group_size=128))
```

```
# TurboQuant
kp, kn, vp, vn = self.turboquant_kv.update(
      input_pos, k, v) # compress to 4-bit

torch.ops.triton.turboquant_sdpa(
  q, kp, kn, vp, vn,
  centroids, rotation, mask  # decompress per-tile
)
```

Fuse directly in the model code

```
# Fuse QKV projections — 3 kernel launches → 1
- self.q_proj = nn.Linear(hidden, q_dim)
- self.k_proj = nn.Linear(hidden, k_dim)
- self.v_proj = nn.Linear(hidden, v_dim)
+ self.qkv_proj = nn.Linear(hidden, q_dim + k_dim + v_dim)
```

Example 2: Kernel-level

Bring in custom performant kernels from other libraries (or LLM generated)

```
from fla.ops.gated_delta_rule.wy_fast import recompute_w_u_fwd_kernel # import from FLA

@triton_op("triton::chunk_gated_delta_rule", mutates_args={})
def chunk_gated_delta_rule(q, k, v, g, beta, state):
    ...
    wrap_triton(recompute_w_u_fwd_kernel)[grid](...) # use directly

# In the model — called like any other torch op
output, state = torch.ops.triton.chunk_gated_delta_rule(q, k, v, g, beta, state)
```

Auto-tuning

```
# Autotune configs for GEMM1 (_fused_moe_kernel).
_GEMM1_CONFIGS = [
    triton.Config({"BLOCK_SIZE_N": 32, "BLOCK_SIZE_K": 32}, num_warps=4,),
    triton.Config({"BLOCK_SIZE_N": 8, "BLOCK_SIZE_K": 256}, num_warps=2,),
]

@triton.autotune(configs=_GEMM1_CONFIGS, key=["N", "K"])
@triton.jit
def _fused_moe_kernel():
```

Here is the “magic” part - ExecuTorch is PyTorch

inference.cpp

```cpp
#include <executorch/extension/llm/runner/text_llm_runner.h>

int main() {}
  auto runner = create_text_llm_runner("model.pte");
  runner->generate("Hello, how are you?");
}
```

Native C++ app, **no libtorch dependency,** can use CUDA

model.pte

exports

model.py

Research/Optimization phase
Stay in Python!

inference.py

- eager
- torch.compile (JIT)
- AOT precompile

Exact same triton kernel in python-based inference loop and native application

Use the power of PyTorch - flexibility and experimentation.

But also get deploy native application easily

Both paths use Inductor underneath the hood

Performance and What's Coming

Metric	Value
Decode speed	~77 tok/s
Model size (bf16)	67 GB
Model size (INT4)	17 GB
Quantization	INT4 weight-only, gs=128, HQQ TurboQuant enabled (Optional), saving 3GB at 200K context length

Measured on A100

- Close the perf gap with SOTA
- CPU offloading – keep only 8 active experts in VRAM, rest in system RAM. Enables RTX 3060/4060 (8-12 GB) deployment
- Cross-platform – Metal (macOS), Windows CUDA, CPU (XNNPACK)

Recap and What's Next

Keep embracing open source AI

What we showed today

Voice – composability, unified cross-platform deployment

Desktop inference – CUDA and Metal via Inductor, custom Triton kernels

MoE – Leverage the power of PyTorch

What's coming

Voice – Expanding to TTS, translation, full-duplex

MoE – CPU offloading, cross-platform (Metal, Windows)

Robotics – proof-of-concept stage

TinyML / Microcontrollers – separate talks at this conference, check them out

One more thing

Open Source Ecosystem so far

14 hardware backends from partners across the industry:
XNNPACK · CUDA · Core ML · Metal · Vulkan · Qualcomm · MediaTek · Arm Ethos-U · Arm Cortex-M · Arm VGF · OpenVINO · NXP · Cadence · Samsung Exynos

Open source ecosystem:
Hugging Face Transformers – export popular models to ExecuTorch
React Native ExecuTorch – declarative toolkit for mobile AI apps
Unsloth – LLM fine-tuning → ExecuTorch deployment pipeline
Ultralytics – YOLO model integration for on-device detection
Arm ML Embedded Evaluation Kit – Cortex-M + Ethos-U deployment
Alif Semiconductor – generative AI on Ensemble MCUs with Ethos-U85
Digica AI SDK – automated PyTorch → edge deployment
...

ExecuTorch is joining the PyTorch Foundation and becoming part of PyTorch Core

ExecuTorch was built through open source collaboration with partners and ecosystem builders from day 1.

Now we're making that official

What this concretely means:

- Increased contributions – maintained by a core set of individual maintainers, welcoming backends, models, and kernels
- Edge AI standardization
- Shared infrastructure – greater sense of ownership from partners

Call To Action

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

Adopt ExecuTorch into your application

- Export your PyTorch models today
- Check out examples models (e.g., LLMs, voice, MoE, Computer Vision…) for reference

Integrate ExecuTorch into library/framework

- Bring multi-backend support to your users

Contribute to ExecuTorch

- Hardware vendors: bring your backend and provide smooth deployment story for PyTorch users
- Model authors: export your model and provide reference implementations
- Kernel developers: contribute performant kernels