DeepLearning.AI PIE & AI truedigital

true

# Practical Edge AI

## From Runtime Choices to Real-World Product Challenges

*Not how to deploy a model once — how to keep it alive on devices you cannot see.*

**Sattaya Singkul (Jo)**
Specialist Lead, AI Solution Engineer · True Digital Group
sattaya.sin@truedigital.com

QUICK FRAMING

# Edge AI in one line

## 1 Latency

Answer in milliseconds, not round-trips.

## 2 Privacy

Sensitive raw data never leaves the device.

***My angle today:*** *the reason to choose edge is usually privacy or cost — not raw speed.*

## 3 Cost

No per-inference cloud bill that scales with users.

## 4 Offline

Works when the network doesn't.

Run inference close to where the data is born — instead of shipping everything to the cloud.

*Source: RunAnywhere, Top Edge AI Solutions 2026; floLIVE Edge AI Best Practices*

# “Edge” is a spectrum, not a place

The same workload sits somewhere on this line — and the model shrinks as you move right.


***The real question isn't “edge or cloud” —*** *it's which stage of your pipeline belongs at which point on this line.*

*Source: Google Cloud Distributed Edge (GDCE) datasheet; Wind River 'Edge AI in the Real World'*

01

THE FRAGMENTED LANDSCAPE

# Runtime choices

*There is no single winner. You pick the engine that matches your hardware envelope.*

# Seven engines, seven envelopes

| Runtime | Best target | Quant. | Edge / strength |
|---|---|---|---|
| **LiteRT (TFLite)** | Android · iOS · Web · IoT | INT8/4 · FP16 | Broad HW delegation, unified NPU API |
| **ExecuTorch** | iOS · Android · macOS | INT8/4 · FP16 | Native PyTorch export, no semantic gap |
| **ONNX Runtime** | Win · Linux · Web (WASM) | INT8/4 · FP32 | One model, server→browser portability |
| **Apple Core AI** | iOS · macOS · visionOS | INT4/8 · FP16 | Neural Engine, zero-copy, stateful KV |
| **TensorRT Edge-LLM** | Jetson · DRIVE | FP8 · NVFP4 · INT4 | C++ real-time, predictable latency |
| **ncnn / MNN** | Mobile ARM CPU/GPU | INT8 · FP16 | Tiny footprint, hand-written assembly |
| **llama.cpp / MLC** | Desktop · local LLM | GGUF INT4/5 | Layer offload, elastic memory |

***Pick by 4 questions:*** *hardware unit (NPU/GPU/CPU?) · source framework · quantization need · binary + RAM budget*

*Source: Google AI Edge (LiteRT); Meta ExecuTorch; Microsoft ONNX Runtime; Apple Core ML; NVIDIA; ncnn/MNN; llama.cpp docs*

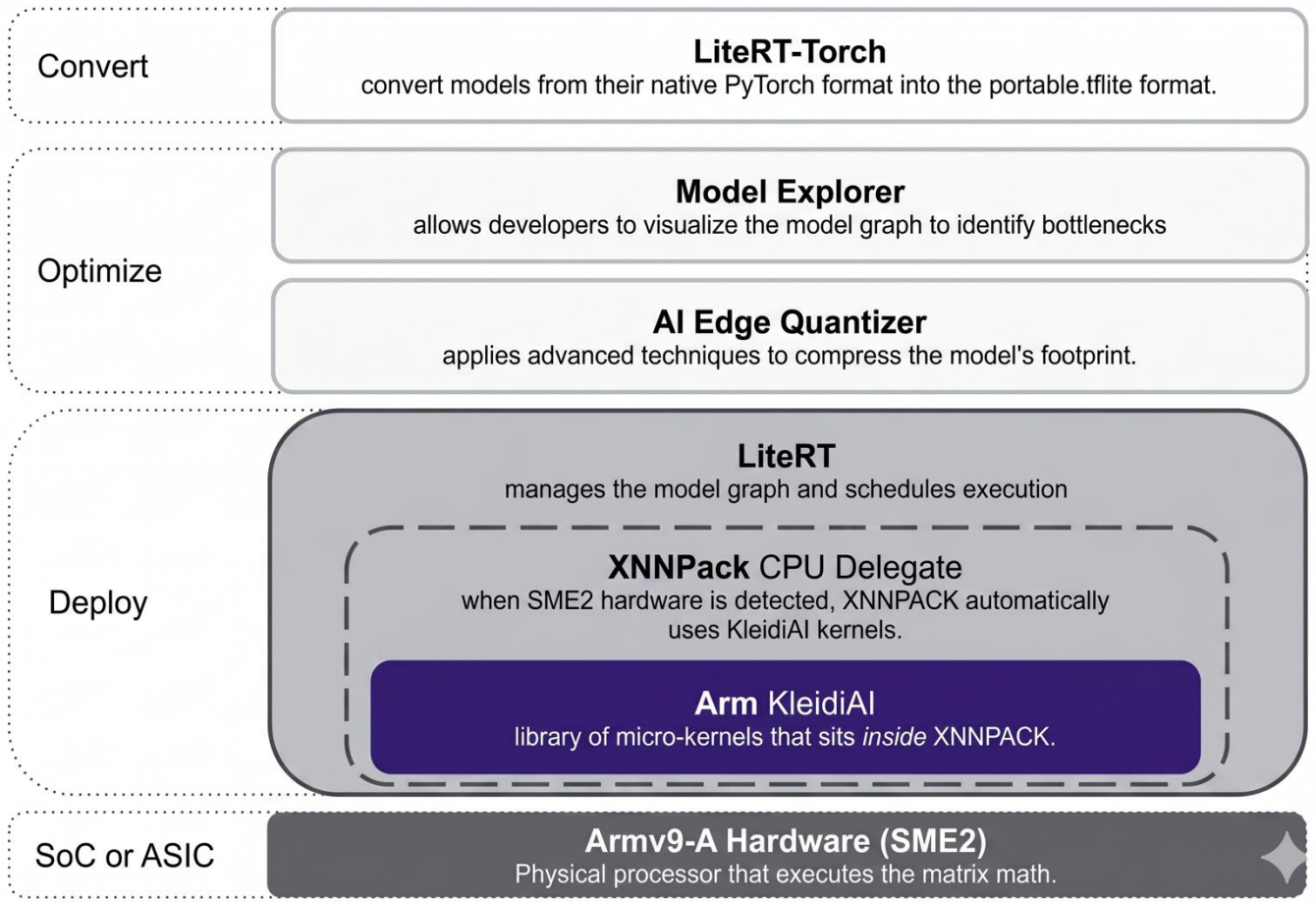
# The challenge: Balancing model quality and mobile reality by LiteRT

**Challenges:**

- **Complexity Gap:** Finding the optimal quantization configuration among many possibilities can be challenging. Furthermore, naively quantizing the entire model's weights yields a severe loss in audio quality.
- **Device Coverage:** Unlocking a path for efficient CPU-based audio generation is more than a technical milestone. It is an opportunity to scale innovative apps across the billions of CPU-powered devices that represent the global smartphone market.

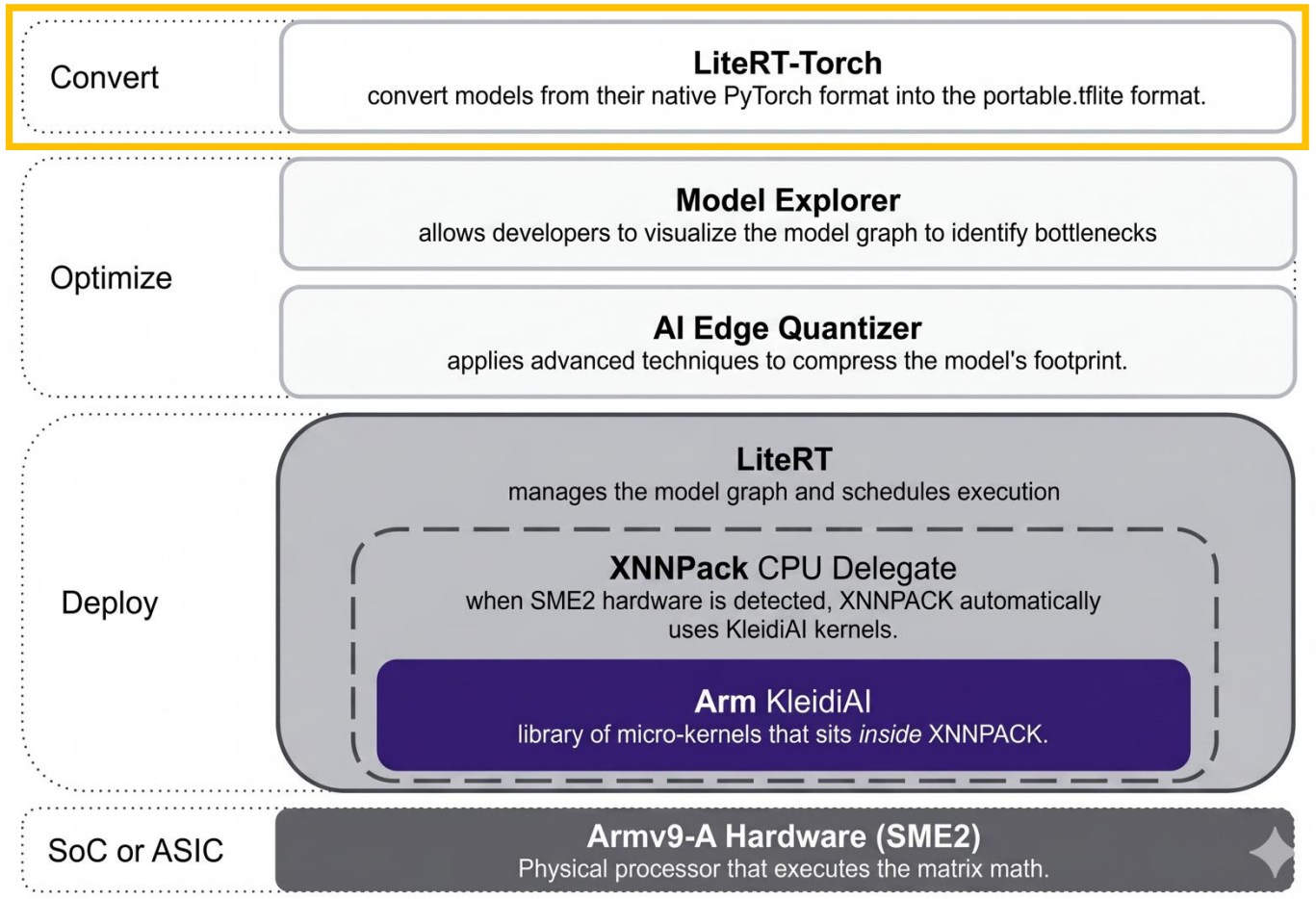
**Google AI Edge: A seamless path from PyTorch to silicon**

A complete end-to-end path with the **Google AI Edge** software stack.




*Source: Google Cloud Blog, 'Benchmark LLMs on-device with AI Edge Portal'; developers.google.com/edge/ai-edge-portal*

# The challenge: Balancing model quality and mobile reality by LiteRT




**Convert**: Convert from PyTorch to .tflite with LiteRT Torch

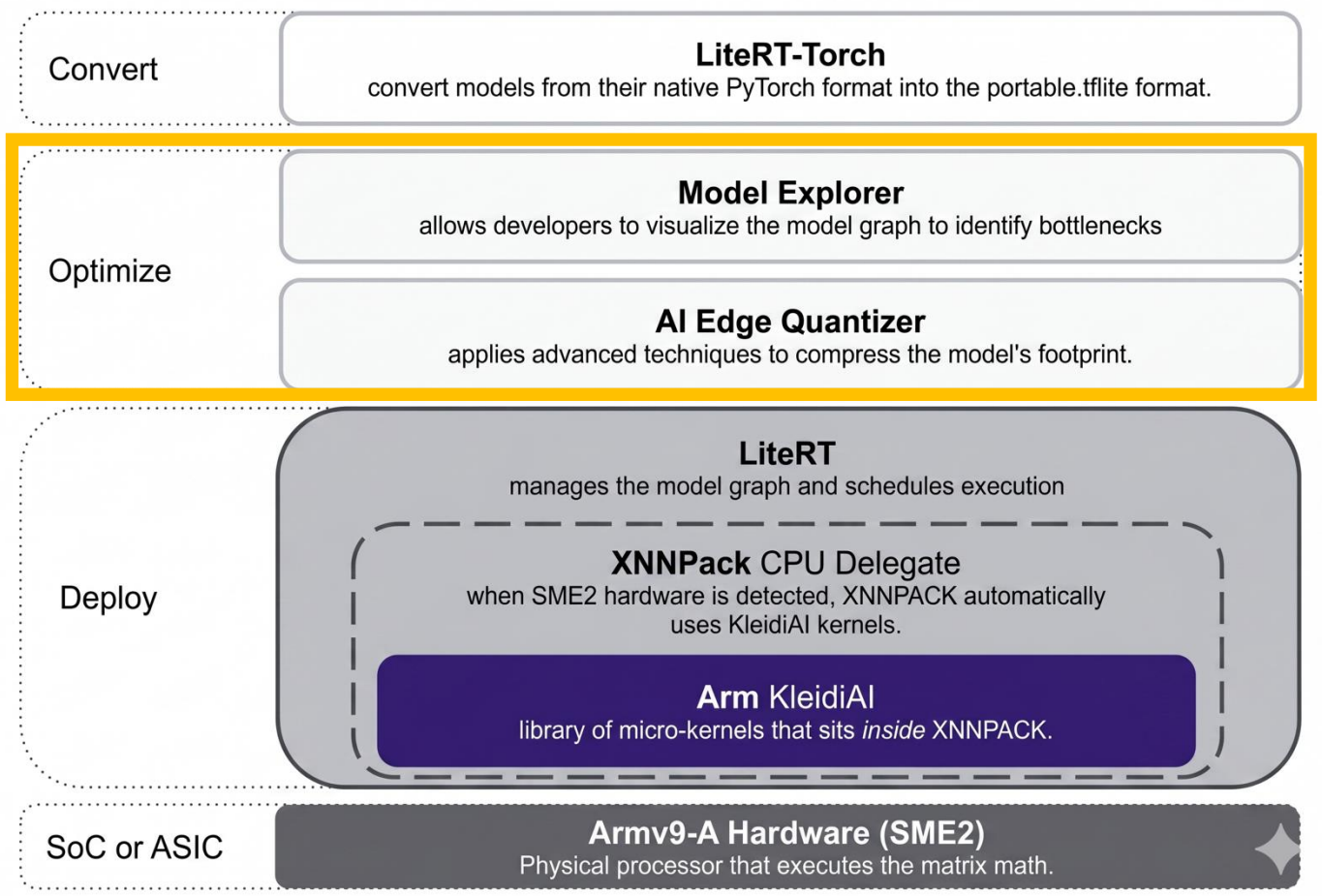
Python

```python
import litert_torch
from litert_torch.quantize import quant_config
from litert_torch.generative.quantize import quant_recipe, quant_recipe_utils


# Specify the quantization format
quant_config_int8 = quant_config.QuantConfig(
        generative_recipe=quant_recipe.GenerativeQuantRecipe(
        default=quant_recipe_utils.create_layer_quant_dynamic(),
    )
)
# Initiate the conversion
edge_model = ai_edge_torch.convert(
    model, example_inputs, quant_config=quant_config_int8
)
```

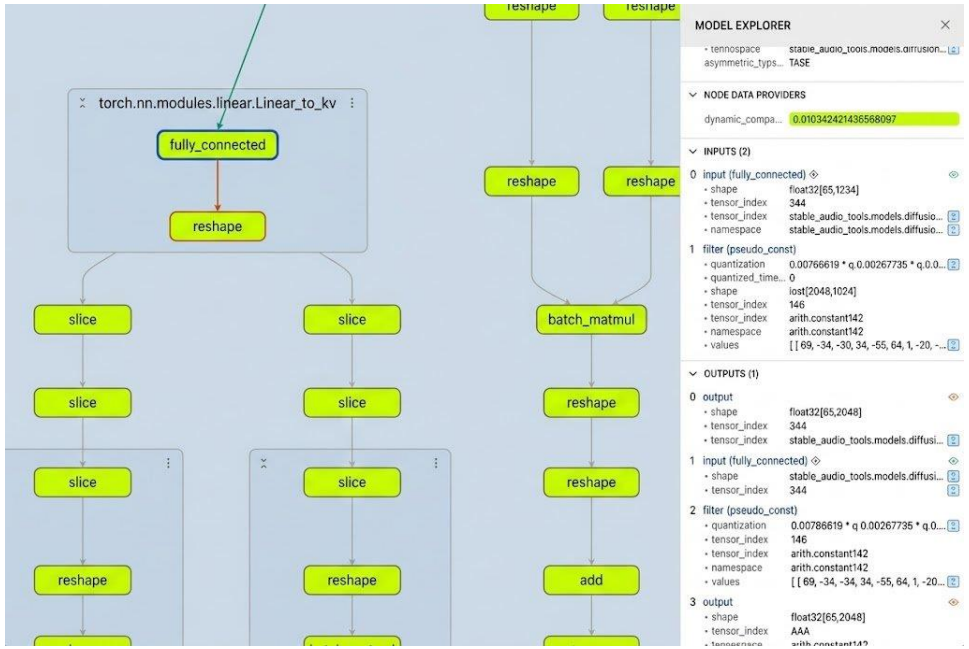
*Source: Google Cloud Blog, 'Benchmark LLMs on-device with AI Edge Portal'; developers.google.com/edge/ai-edge-portal*

# The challenge: Balancing model quality and mobile reality by LiteRT




**Optimize:** Optimize with Model Explorer and AI Edge Quantizer

All AI layers are green (visualized by **Model Explorer**), indicating low error values (FP32 vs. FP32+INT8).

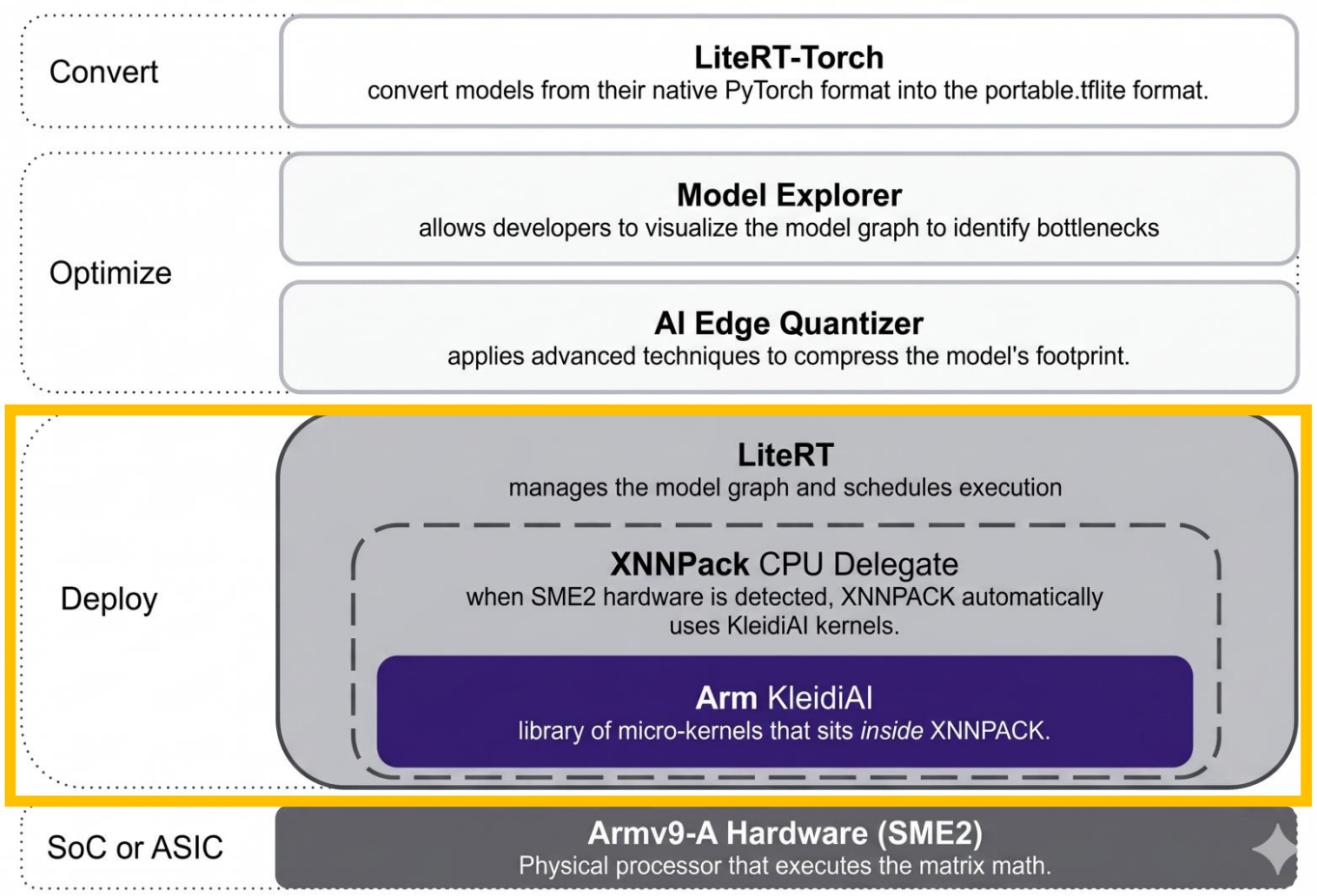



Fully connected layer with INT8 dynamic quantization. The error rate, reported under "NODE DATA PROVIDERS," is approximately 1%.

Once the suitability of INT8 quantization was confirmed, we utilized the **AI Edge Quantizer** to optimize the model from FP32 to INT8.

*Source: Google Cloud Blog, 'Benchmark LLMs on-device with AI Edge Portal'; developers.google.com/edge/ai-edge-portal*

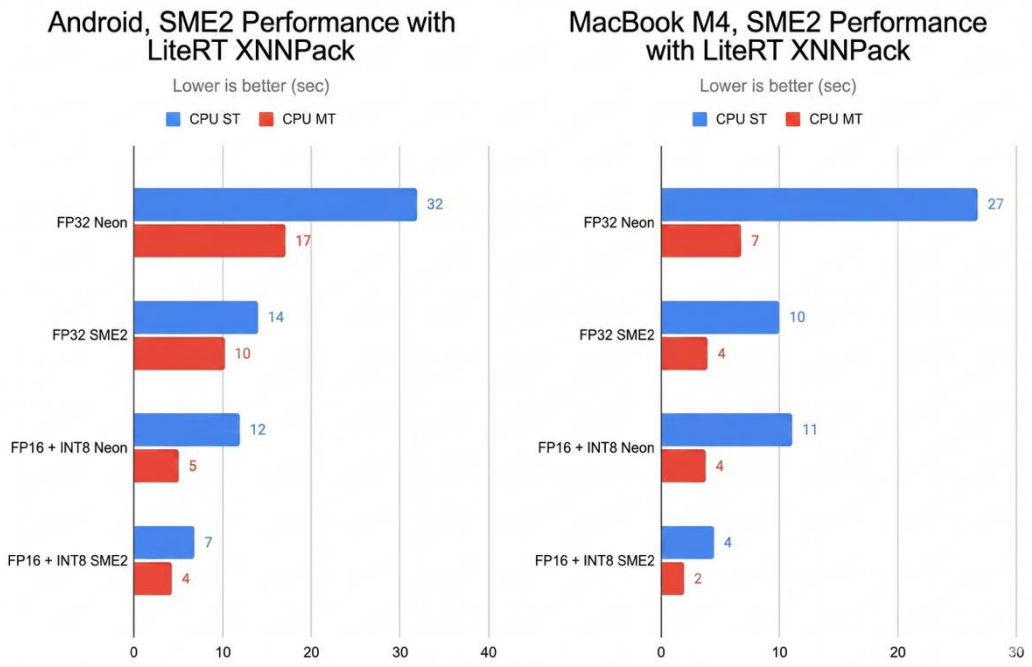
# The challenge: Balancing model quality and mobile reality by LiteRT




**Deploy:** High-performance inference with LiteRT via XNNPack & KleidiAI

**LiteRT** on an Android mobile device, it defaults to the **XNNPACK** delegate with automatic optimization for CPU inference.
(XNNPACK integrates KleidiAI directly within the latest LiteRT API)

**Results**: Faster, smaller, and high-quality audio generation with a lower footprint




*Source: Google Cloud Blog, 'Benchmark LLMs on-device with AI Edge Portal'; developers.google.com/edge/ai-edge-portal*

SAME MODEL, VERY DIFFERENT MACHINES

# Gemma-4 E2B on LiteRT-LM

Mixed 2/4/8-bit, GPU path — decode throughput & peak RAM tell the real story.

**S26 Ultra**

52

tok/s decode

**676 MB RAM**

Snapdragon NPU

**iPhone 17 Pro**

56

tok/s decode

**1,450 MB RAM**

A19 Pro / Metal

**MacBook M4**

160

tok/s decode

**1,623 MB RAM**

Apple M4 GPU

**Raspberry Pi 5**

8

tok/s decode

**1,546 MB RAM**

Broadcom CPU only

*20× spread in throughput across devices in one fleet — your slowest target defines the design.*

*Source: Google Developers Blog, 'Blazing fast on-device GenAI with LiteRT-LM' (Gemma-4 E2B benchmarks)*

# Running an LLM on a phone, honestly

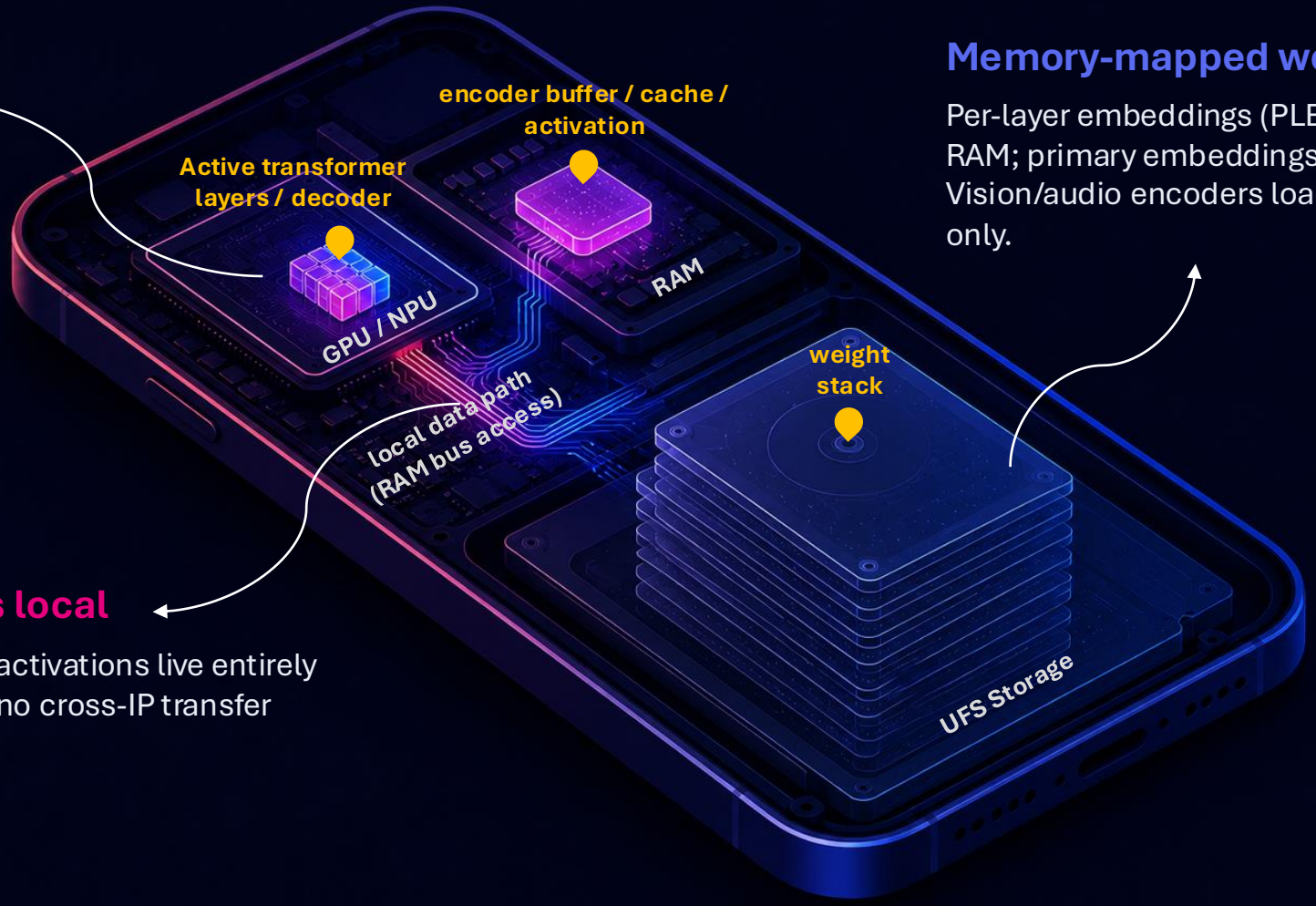
LiteRT-LM is the orchestration layer that makes on-device GenAI fit in memory and feel fast.

## Multi-Token Prediction

Target model + lightweight draft model on the same IP block (GPU) → speculative decoding speedup on mobile.

## Memory-mapped weights

Per-layer embeddings (PLEs) kept out of RAM; primary embeddings mmap'd. Vision/audio encoders load on demand only.




## KV cache stays local

Shared KV cache + activations live entirely in local memory — no cross-IP transfer penalty.

*Net effect: Gemma-class models run on flagship phones with a working set in the hundreds of MB — not gigabytes.*

*Source: Google Developers Blog, 'Blazing fast on-device GenAI with LiteRT-LM'; LiteRT-LM Overview, Google AI Edge*

DeepLearning.AI PIE & AI truedigital

# Speculative decoding, drawn out

**6 passes → 6 tokens**

STANDARD DECODING


*one forward pass = one token*

A small draft model guesses several tokens; the big model verifies them all in one pass. Accepted guesses are free speed.

SPECULATIVE DECODING (MTP)


*draft guesses 4 → target verifies all at once*

**1 pass → 3 tokens**

3 accepted + 1 corrected — then repeat. Fewer big-model passes = real wall-clock speedup on a phone GPU.

**TARGET model — one verification pass**

*Source: Google Developers Blog, 'Blazing fast on-device GenAI with LiteRT-LM' (Multi-Token Prediction)*

# Benchmark on real devices, not your laptop

AI Edge Portal runs your model across a large fleet of physical devices in the cloud — the number that matters is the one from the slowest target you actually ship to.

## WHAT YOU GET

- **Real-device matrix**
  latency, memory & init time across many SoCs
- **Per-accelerator view**
  CPU vs GPU vs NPU on the same model
- **Pre-ship confidence**
  catch the slow/expensive device before users do
- **Feeds your rollout**
  baseline numbers for canary SLOs & rollback gates

## THIS CLOSES THE LOOP

Benchmarking isn't a one-time pre-launch step — it's the baseline your lifecycle measures against.

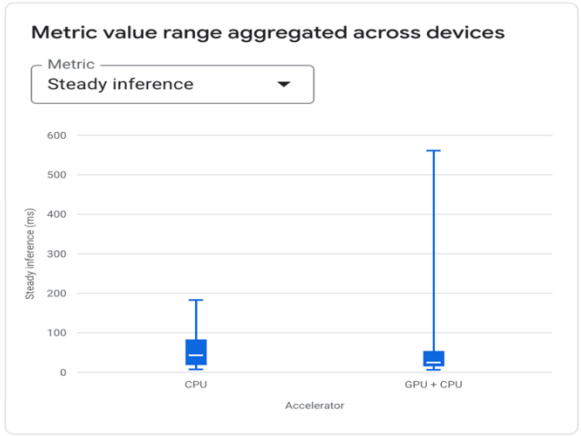
- Benchmark fleet
- Set per-device SLOs
- Canary against baseline
- Regress → rollback

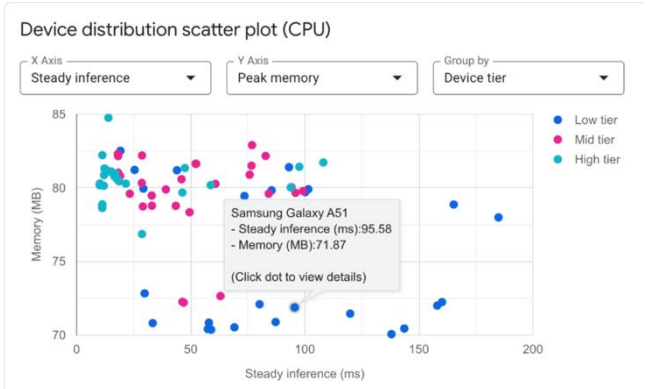
*Source: Google Cloud Blog, 'Benchmark LLMs on-device with AI Edge Portal'; developers.google.com/edge/ai-edge-portal*

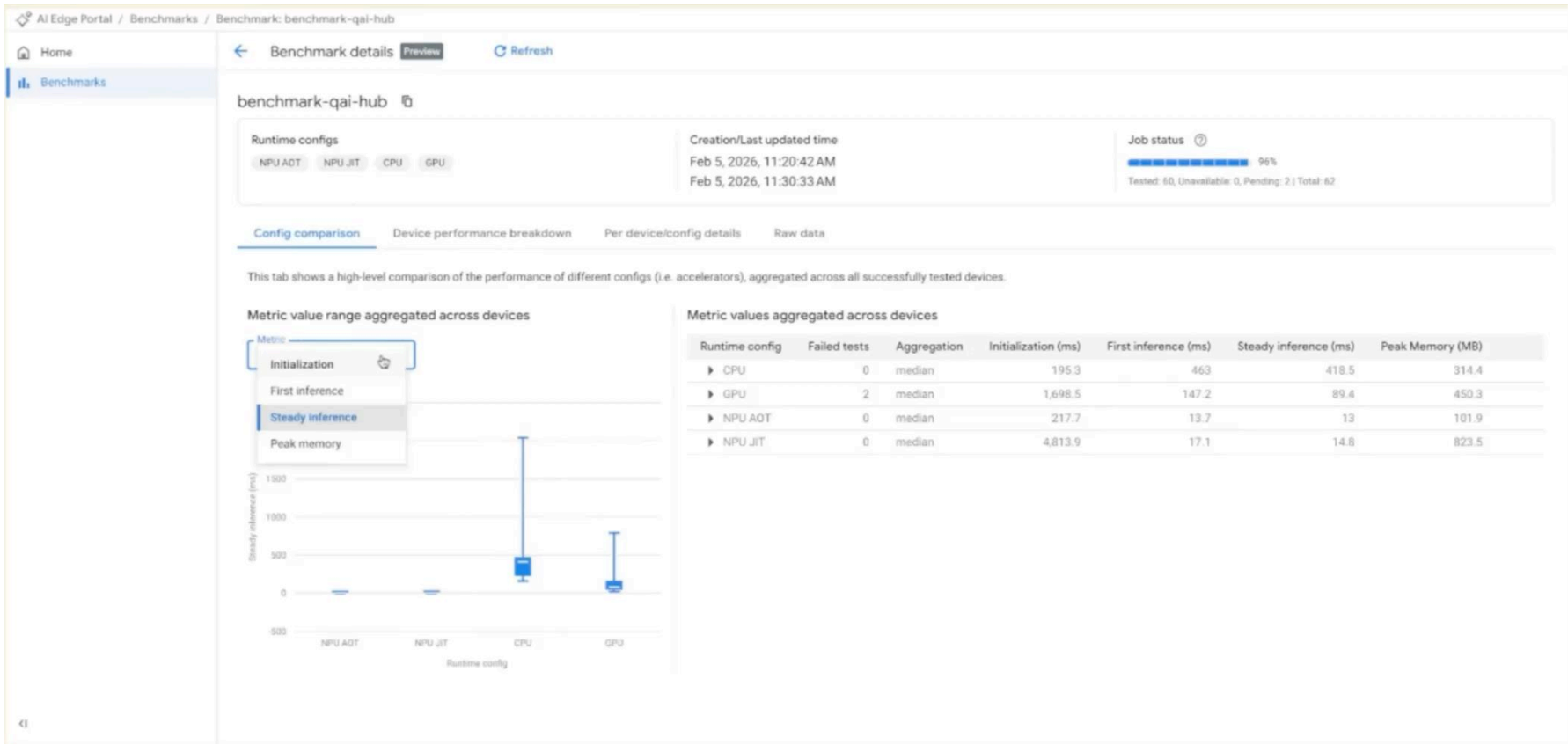
# Benchmark on real devices, not your laptop

**Google AI Edge Portal** provides a benchmark service across 100+ of the most popular mobile phones with insights on ML workloads across devices, accelerators and configurations. Developers can now make data-driven deployment decisions, such as whether to use Ahead-of-Time (AOT) or Just-in-Time (JIT), that best suit their use cases and their target devices.

Register for Preview Access




*Source: Google Cloud Blog, 'Benchmark LLMs on-device with AI Edge Portal'; developers.google.com/edge/ai-edge-portal*

02

EDGE VS CLOUD IS A SPECTRUM

# Partitioning & privacy

*Not "all edge or all cloud" — decide piece by piece, and put privacy in the architecture.*

# Decide each stage on four axes

**1** **Raw data sensitive?**
Yes → run on edge

**2** **Latency you can feel?**
Yes → run on edge

**3** **Model small enough?**
Small → run on edge

**4** **High call frequency?**
High → edge preprocess / cascade

## A VOICE PIPELINE, MAPPED

| | |
|---|---|
| Wake-word | EDGE |
| Speech enhancement | EDGE |
| Speech-to-text | EDGE |
| PII masking | EDGE |
| Reasoning / agent | CLOUD |
| Text-to-speech | EDGE / CLOUD |

*Always-on + most-sensitive stays local. Heavy reasoning earns its trip to the cloud.*

Source: Splitwise, arXiv:2512.23310; Two-Stage Cascade Architecture, Emergent Mind; UCSD Unified Splitting

THE IDEA WORTH STEALING

# Privacy by architecture, not privacy by policy.

*The more sensitive the raw data, the closer to the source you finish processing it — then send only what's safe.*

*"Hi, this is John, my card is 4912 3456..."* → **"Hi, this is [NAME], my card is [CARD]"**

**01**

**Raw audio**

Stays on device. Ring buffer, never crosses the network.

→

**02**

**On-device STT + PII mask**

Names, numbers, bank details replaced locally before anything leaves.

→

**03**

**Sanitized text → cloud**

Only safe, structured payload leaves the device.

Satisfies GDPR Art. 25 & data minimisation by design — and makes "right to erasure" almost trivial.

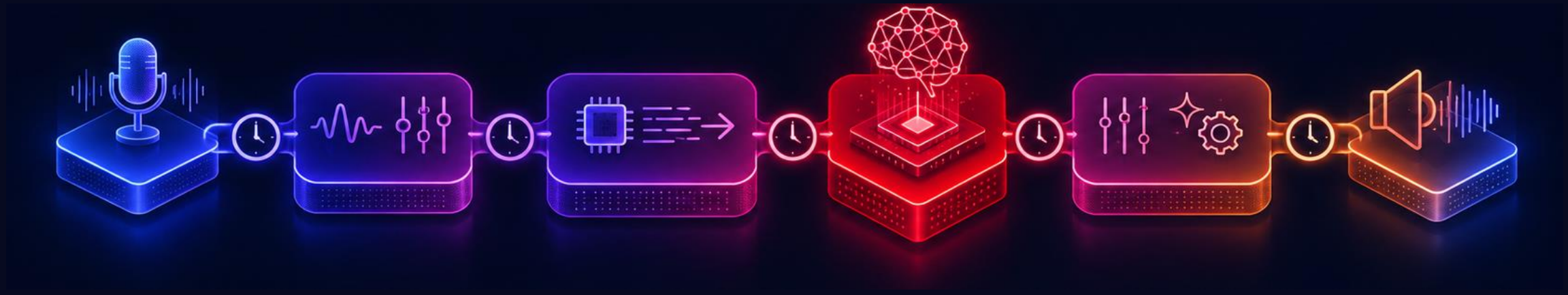
*Same pattern behind on-edge call-center STT under compliance constraints.*

*Source: GDPR Art. 25 & 5(1)(c), EU Commission; OpenAI Privacy Filter; K2view GDPR data masking*

THE NUMBER YOU FORGOT TO MEASURE

# “Inference time” is not latency

Users feel the whole chain. The model is often the smallest slice of it.



| 12% | 15% | 18% | 28% | 12% | 15% |
|---|---|---|---|---|---|
| Audio capture & buffering | Pre-process + feature extract | Memory transfer CPU↔NPU | Model inference | Post-process + decode | Output / TTS / render |

↑ *the part most people optimize*

**And the ones that ambush you in production:**

### First-inference warm-up

cold model load can be 10–100× the steady-state pass

### Thermal throttling

sustained load → clock drops → latency creeps up over minutes

### Memory pressure

GC / paging stalls when other apps fight for RAM

*Source: Wind River 'Edge AI in the Real World'; Google AI Edge LiteRT performance guidance*

# A demo ends at “it runs.” A product starts there.

## Devices you can't see

Thousands of phones, speakers, gateways — no SSH, no logs, no debugger.

## Failures you can't reach

When it breaks on a customer's device, you can't just redeploy and watch.

## Models that keep changing

New weights, new knowledge, new versions — all on different release cycles.

***So you can't improvise.*** *All three force one thing — a lifecycle you designed up front.*

*Source: Wind River, 'Edge AI in the Real World'; Milvus, edge AI updates reference*

# Let's talk.

*Questions on runtimes, quantization, on-device RAG, or anything from the lifecycle loop.*

**Sattaya Singkul (Jo)**
Specialist Lead, AI Solution Engineer · True Digital Group