AGENTIC INFERENCE ENGINEERING · EVIDENCE-DRIVEN OPTIMIZATION

From PyTorch to the Edge:
Agentic Synthesis of Inference Runtimes for Heterogeneous Hardware

Thomas Cottenier · Arm

arm

© 2026 Arm

Outline

LOOM · PRESENTATION OUTLINE

Motivation	The runtime gap and the cost of specialization
Introducing Loom	Agentic runtime generation and measurement
Qwen3 + Gemma 4 experiments	Quality and performance on Apple M4 Max
Generalization	Raspberry Pi, upcoming phone and DGX Spark
Conclusions	Results and the economics of bespoke runtimes
Q&A	Questions and discussion

© 2026 Arm

LOOM · CONTEXT

Motivation

The runtime gap · the diversity of targets
The case for autonomous optimization

arm

© 2026 Arm

An Arm system runs everyone else's ML runtime

RASPBERRY PI 5 · RUNTIME LAYER

Raspberry Pi 5 uses Broadcom BCM2712 with four 64-bit Arm Cortex-A76 CPU cores. Yet no Arm first-party deployment runtime appears in this YOLO26n comparison.

ARM PLATFORM

Raspberry Pi 5

BCM2712 · 4× Cortex-A76 · CPU inference

OpenVINO	Intel	**ONNX Runtime**	Microsoft
LiteRT	Google	**ExecuTorch**	Meta
NCNN	Tencent	**MNN**	Alibaba

The gap

Even Intel provides an optimized ML path for this Arm device. Arm supplies the architecture and compute IP, but no Arm first-party ML runtime is represented in this benchmark.

The strategic gap: Arm owns the compute architecture, while other vendors own the runtime experience presented to developers.

YOLO26n · 640 px · Raspberry Pi 5

Lower latency / higher throughput is better

RUNTIME	MS/IMAGE	IMAGES/S	MAP50–95
NCNN	71.56	13.97	0.4784
MNN	97.40	10.27	~0.475
OpenVINO	109.65	9.12	0.4734
ONNX Runtime	137.98	7.25	0.4734
ExecuTorch	142.49	7.02	0.4772
LiteRT	257.00	3.89	~0.473
*PyTorch FP32**	*299.40**	*3.34**	*0.4760**

ARM HARDWARE

Third-party runtime layer

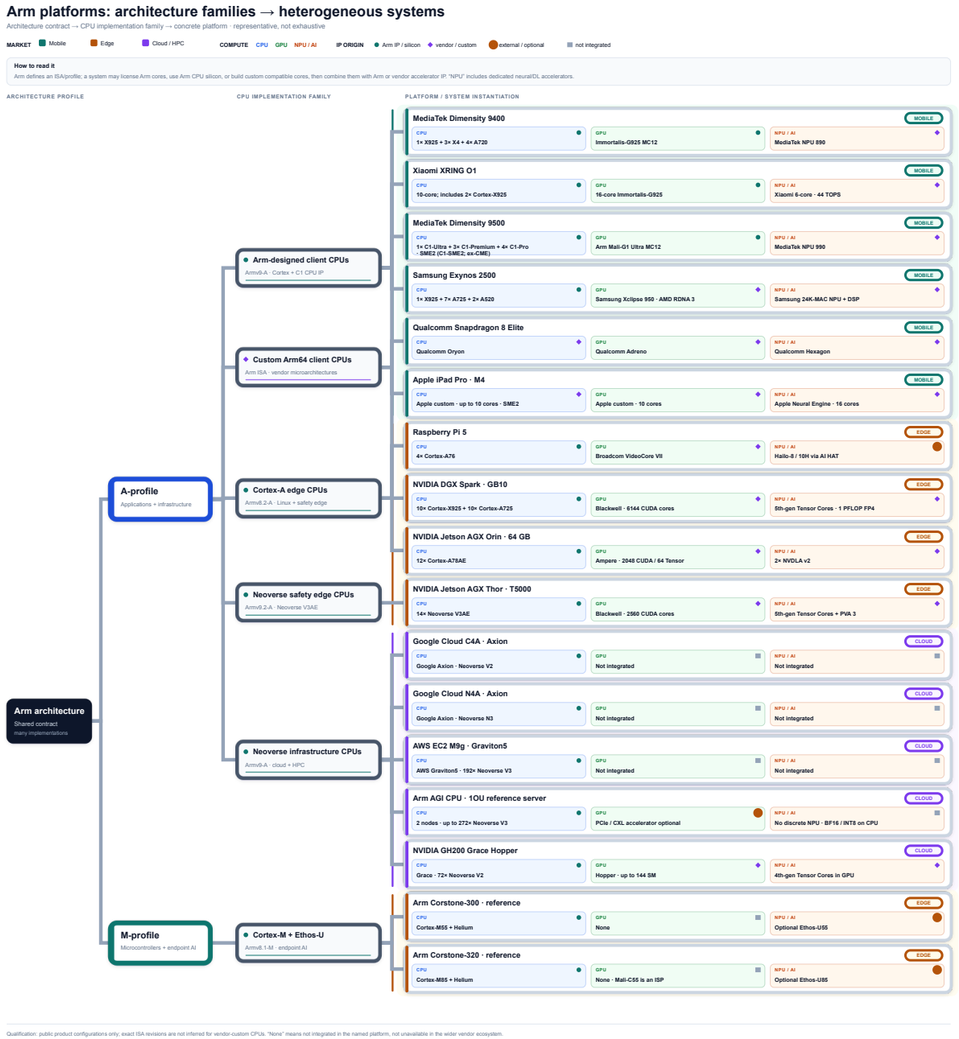
Intel · Microsoft · Google · Meta · Tencent · Alibaba

arm

© 2026 Arm

One architecture contract, many concrete systems

ARM PLATFORM LANDSCAPE · INTERACTIVE



© 2026 Arm

The unit of work is a target, not an architecture

WHY HAND-TUNING DOES NOT SCALE

100s **of SoC designs across dozens of partners**

Different CPUs, GPUs and NPUs.
Different memory, compilers, drivers and power envelopes.

THE CONVENTIONAL PATH

Extend the runtime ecosystem

PyTorch	ExecuTorch	LiteRT	ONNX Runtime
llama.cpp	MNN	NCNN	

Frameworks × targets × CPU / GPU / NPU paths

- Add backends, lowering and optimized kernels
- Integrate compiler and driver differences
- Revalidate models, quality and performance
- Upstream and maintain across releases

THE ECONOMIC CONSTRAINT

Per-target optimization costs engineer-years

- Each target combines a different memory hierarchy, compiler, driver and power envelope.
- A bespoke human team for every combination has never been economically viable.
- Shared runtime capability is valuable; repeated optimization for the long tail is hard to justify.

The opportunity Make tuning economical where a dedicated human team would never exist.

© 2026 Arm

LOOM · THE APPROACH

Introducing Loom

An agentic harness for building and optimizing
a runtime for each model and hardware target.

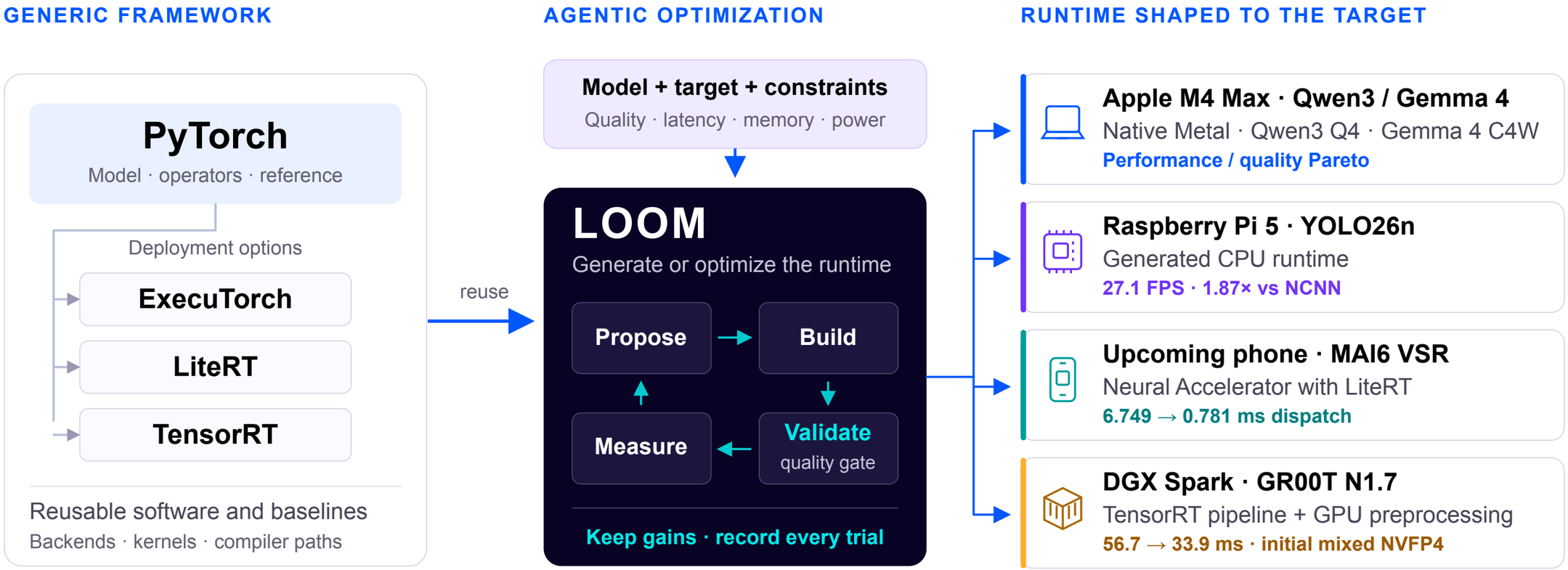
arm

© 2026 Arm

Project Loom: a bespoke runtime for each target

THE BET

Run a durable search that composes PyTorch, ExecuTorch, TorchAO, and target-specific compilers into a runtime that satisfies that target's constraints.



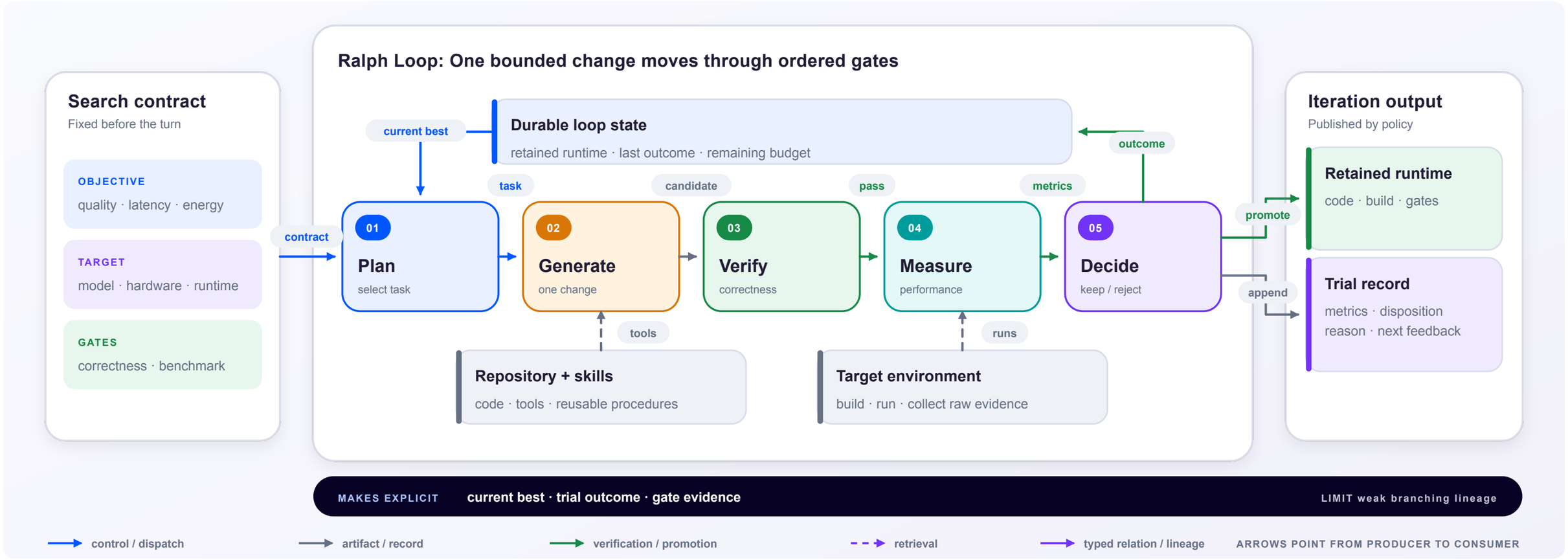
One method. Different artifacts. Correctness → quality → target performance → retained evidence

© 2026 Arm

An agentic optimization harness running on AWS

WHAT IS LOOM?

Loom **generates or optimizes an inference runtime for a specific model on a specific hardware platform**. It makes that work autonomous, verifiable, and reproducible—so the same evidence-driven loop can scale across inference problems and platforms.



Agentic harness powered by **OpenAI gpt-5.6-Sol and gpt-6.1-Sol**.

© 2026 Arm

LOOM · APPLE SILICON EXPERIMENTS

Qwen3-4B
Gemma4-E4B

Reuse weights to study execution.
Build calibrated weights to study quality and execution.

arm

© 2026 Arm

Goal: approach the quality–performance frontier

APPLE M4 MAX · QWEN3 AND GEMMA 4 EXPERIMENTS

Show that an **agent-generated runtime** can meet the quality–performance Pareto frontier of highly optimized general-purpose runtimes: **llama.cpp and MLX**.

Experiment	Qwen3-4B · reuse existing Q4	Gemma 4 E4B · own C4W format
Demonstrate	Preserve an existing Q4 model's behavior while optimizing its runtime.	Design C4W, calibrate it, then optimize a native runtime.
Weights / calibration	mlx-community/Qwen3-4B-4bit ↗ Reuse Q4 g64 unchanged; no recalibration.	google/gemma-4-E4B-it ↗ (BF16) Row-wise C4W; GPTQ-style C4 calibration.
Native implementation	Objective-C++ / Metal: batched prefill and resident K/V decode.	Metal C4W kernels, FP16 per-layer embeddings, shared K/V and buffer reuse.
Quality evidence	Direct KLD vs MLX checks runtime fidelity. BF16 KLD includes inherited Q4 loss.	C4 PPL and KLD vs BF16; compare with MLX Q4 and llama.cpp.
Performance evidence	Native vs MLX-LM: matched Q4 weights, prefill and generated decode.	Native C4W prefill and cached decode. Baseline runtimes use their respective formats.

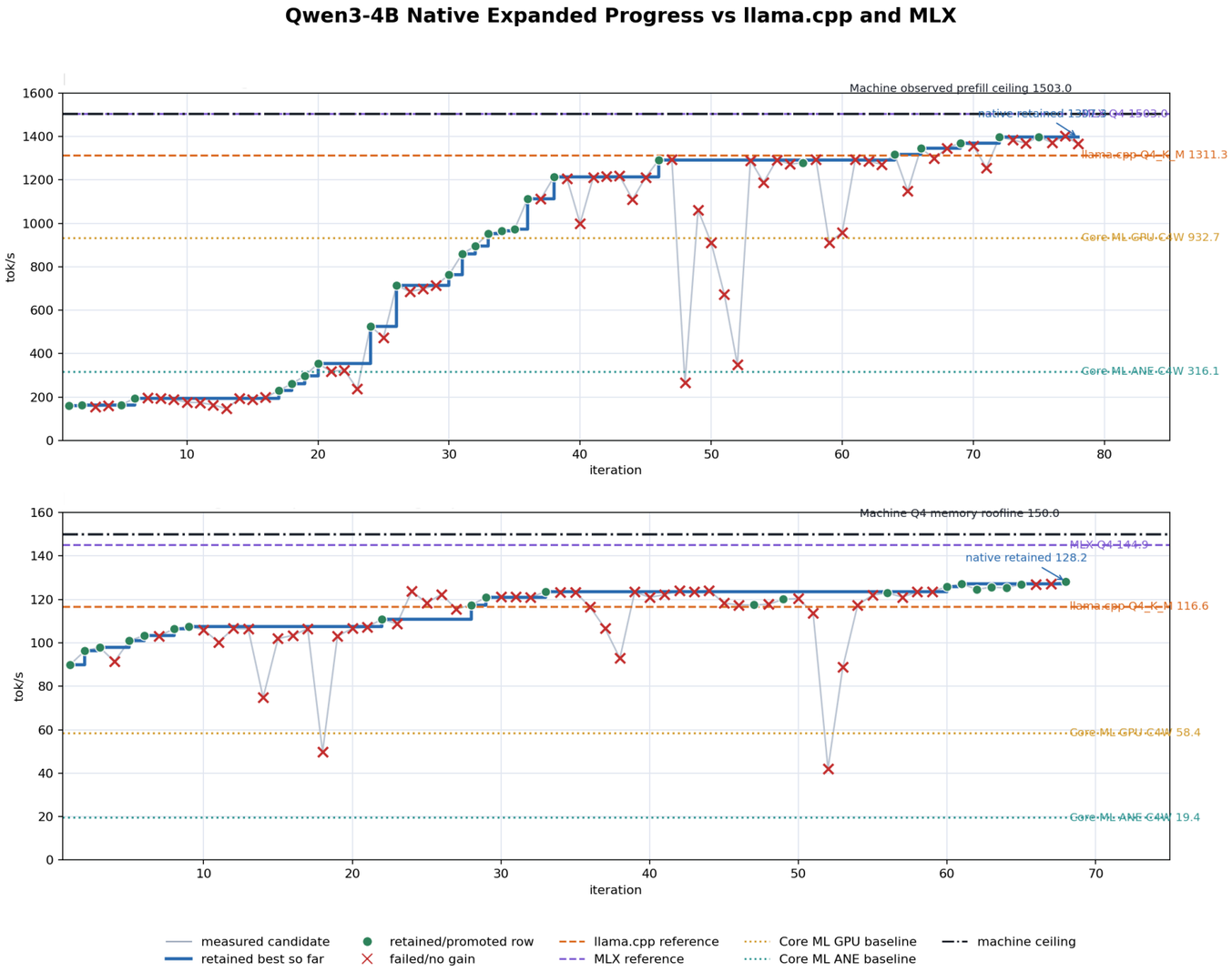
Read quality and speed together. Qwen reuses MLX Q4 weights. Gemma uses C4W; MLX and llama.cpp baselines use their own formats.

arm

© 2026 Arm

The ratchet chart is one view of the experiment DAG

DAG PROJECTION · APPLE SILICON M4 · QWEN3-4B



Trial node + builds_on edge

Every point is an immutable experiment; its predecessor edge preserves the branch hidden by the time axis.

Correctness — build, test, smoke

The candidate must compile, pass focused tests, run end to end, and preserve an expected output—such as decode tail token 804.

Quality — PPL/KLD vs BF16

Measure perplexity, mean / p99.9 KL, argmax agreement, and top-5 overlap. Checked-in 3,069-token MLX Q4 smoke: PPL 15.13 vs 12.96; mean KL 0.234; argmax 80.8%; top-5 78.1%.

Performance — matched tok/s

Measure the same workload and device: exact3072 prefill or seq1024 decode, candidate versus retained control.

Decision — retain, reject, promote

Failed trials remain nodes; only a quality-valid speed win advances the blue retained-head staircase.

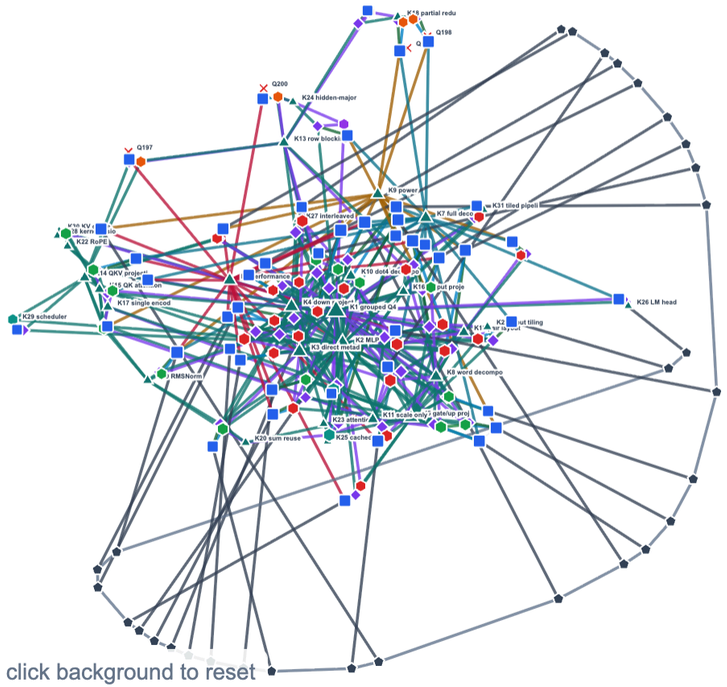
© 2026 Arm

From implicit claims to connected evidence

TYPED GRAPH · LIVE REPOSITORY VIEW

Artifact-derived semantic overlay · Apple Silicon Metal · Qwen3-4B decode

200 entities · 623 auditable relationships · average degree 6.2 · 90% semantic · click a node to inspect its neighborhood



Drag to pan · wheel to zoom · hover for details · click background to reset

Search titles or IDs

NODE TYPES

Source 45
Imported report or primary evidence document

Claim 45
Proposition extracted from the evidence

Decision 45
Outcome node; fill color denotes promotion status below

Concept 31
Canonical domain entity extracted from source artifacts

Commit 30
Git revision that introduced one or more source artifacts

Open Question 4
Unresolved blocker or follow-up

Decision color

promoted · retained · rejected · blocked · superseded

EVIDENCE CHAINS 45

© 2026 Arm

Qwen3-4B: data-format tradeoffs

APPLE M4 MAX · C4 · 10,230 TARGETS · DATA FORMATS

MLX Q4 / Q8
Affine codes + group scales and biases

GGUF Q4 / Q8
Block scales; K_M mixes precisions

Runtime / format	bpw†	KLD ↓	PPL ↓	Prefill 1,024 tok tok/s ↑	Decode 256 steps tok/s ↑
BF16 reference	16.00	0	19.744	—	—
llama.cpp Q8_0	8.50	0.0004	19.894	974.5	55.347
llama.cpp Q4_K_S	4.50	0.0607	20.817	1,379.6	135.154
llama.cpp Q4_1	5.00	0.0699	21.455	1,299.5	81.355
llama.cpp Q4_0	4.50	0.0757	20.799	1,448.2	116.415
llama.cpp Q4_K_M	4.5+	0.0953	22.161	1,584.5	129.538
MLX Q8	8.50	0.0972	20.740	1,567.1	93.604
MLX Q4 g32	5.00	0.1874	22.291	1,595.4	140.341
MLX Q4 g64	4.50	0.2213	22.509	1,735.5	150.027
Native Q4 g64	**4.50**	**0.2217**	**22.476**	**1,792.8**	**160.147**
Initial Native Q4 g64	**4.50**	**0.2217**	**22.476**	**92.8**	**67.134**

† Nominal quantized-matrix bits/weight, including scales and biases; “+” denotes mixed precision. Dense tensors excluded.

Read the tradeoff

- Lower KLD / PPL means higher quality.
- Lower bpw reduces weight traffic.
- Native reuses MLX Q4 g64 weights without recalibration.
- Initial Q4 shares the Native Q4 weight-format KLD / PPL.

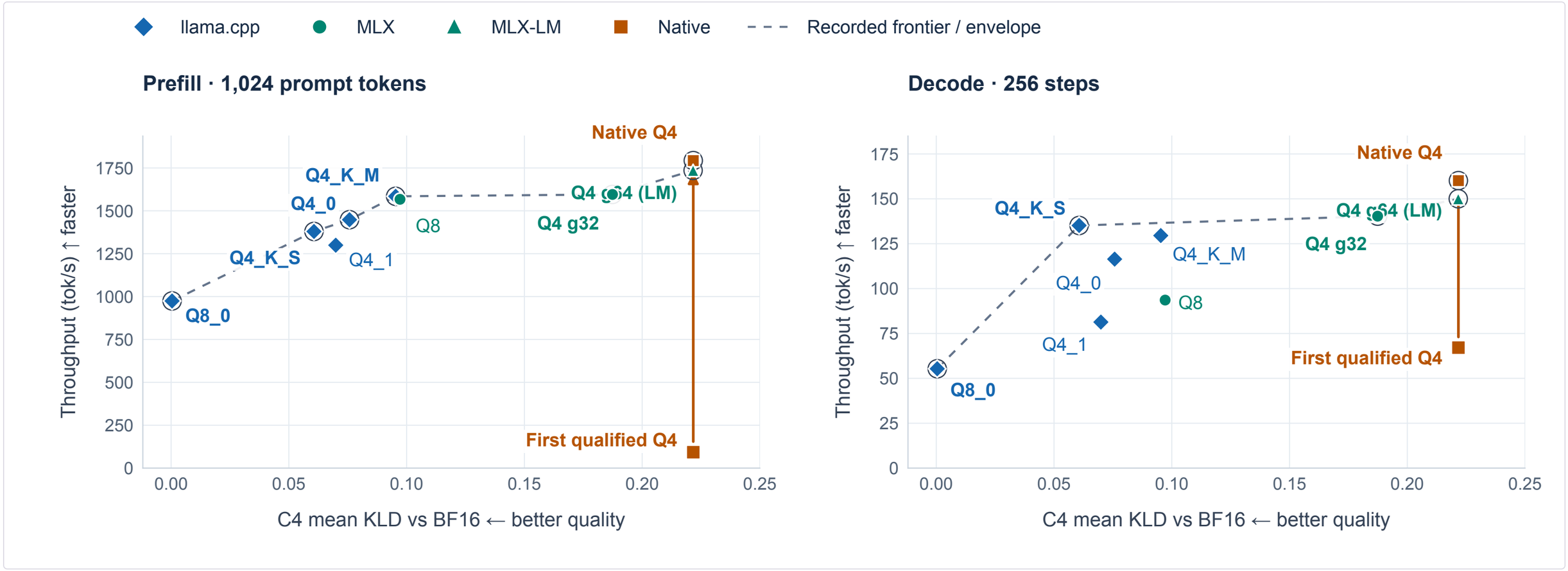
arm

© 2026 Arm

Qwen: format curves set the performance context

QWEN3-4B · FORMAT SURVEY · KLD VS BF16

Native Q4 and MLX-LM: 1,024 prompt tokens · 256 decode steps.



© 2026 Arm

Qwen: generated native code

QWEN3-4B · NATIVE RUNTIME · ORIGINAL MLX Q4 G64

25 Metal kernels · 3,829 Metal lines · 358 host + 47 ABI lines.

Objective-C++ · one prefill layer · runtime.mm lines 159–179

```
void prefill_layer(Command command, Params p, uint32_t layer, uint32_t count) {
  p.layer_start = p.kv_layer_start = layer;
  const uint32_t tiles = (count + 31) / 32;
  encode(command, NormBatch, MTLSizeMake(count, 1, 1), 256, p, {p_hidden, input_norm, p_norm}, 3);
  encode(command, TiledQkv, MTLSizeMake(192, tiles, 1), 128, p,
         {p_norm, q.words, k.words, v.words, q.scales, k.scales, v.scales,
          q.biases, k.biases, v.biases, p_qraw, p_kraw, p_vraw}, 13);
  encode(command, RopeBatch, MTLSizeMake(40, count, 1), 256, p,
         {p_qraw, p_kraw, q_norm, k_norm, cos_table, sin_table, p_q, p_k}, 8);
  encode(command, WriteKV, MTLSizeMake(8, count, 1), 128, p, {cache_k, cache_v, p_k, p_vraw}, 4);
  encode(command, count > 8 ? MatrixAttention : SmallAttention,
         count > 8 ? MTLSizeMake(tiles, 32, 1) : MTLSizeMake(8, count, 1), 128, p,
         {p_q, cache_k, cache_v, p_attention}, 4);
  encode(command, TiledOutput, MTLSizeMake(80, tiles, 1), 128, p,
         {p_attention, p_hidden, o.words, o.scales, o.biases, p_projected}, 6);
  encode(command, PostNormBatch, MTLSizeMake(count, 1, 1), 256, p, {p_projected, post_norm, p_norm}, 3);
  encode(command, TiledGateUp, MTLSizeMake(304, tiles, 1), 128, p,
         {p_norm, gate.words, up.words, gate.scales, up.scales, gate.biases, up.biases, p_intermediate}, 8);
  encode(command, TiledDown, MTLSizeMake(80, tiles, 1), 128, p,
         {p_projected, p_intermediate, down.words, down.scales, down.biases, p_hidden}, 6);
}
```

25 METAL KERNEL ENTRYPOINTS

1 MLX-adapted · 24 generated

Generated entrypoints also use MLX-derived math helpers.

MLX CODE REUSE

~50% of code reused from MLX

~2,050 / 4,234 lines; ~54% of Metal. Includes copied + adapted code.

Browse host source ↗

Browse Metal kernels ↗

arm

© 2026 Arm

Gemma: why build a native C4W runtime?

GEMMA 4 E4B · EXPERIMENT MOTIVATION

ORIGINAL EXPORT FLOW **PyTorch → TorchAO C4W → ExecuTorch → Core ML**

THE OPPORTUNITY

C4W minimizes weight traffic

- **4.005 bpw:** versus 4.50 for MLX Q4 g64 / Q4_0 and 5.00 for Q4_1.
- **Row-wise INT4:** one FP16 scale per output row; less metadata than grouped Q4.
- **GPTQ-style calibration on C4:** fit INT4 codes + row scales to improve quality.

THE OBSERVED GAP

Decode pays repeated runtime costs

- **ExecuTorch / Core ML:** split delegation; KV updates outside the delegate.
- **Direct Core ML C4W:** 11.60 tok/s decode (128 prompt tokens + 64 updates).
- **Per-token costs:** 44 synchronous calls, host KV copies, CPU softcap + argmax.

THE EXPERIMENT

Have Loom close the gap autonomously

- Generate an Apple Silicon runtime: packed C4W kernels, fusion, GPU-resident KV and lean dispatch.
- Beat our measured Core ML paths; close the **quality–performance gap with llama.cpp and MLX.**

© 2026 Arm

Gemma4-E4B: data-format tradeoffs

APPLE M4 MAX · C4 · 4,094 TARGETS · DATA FORMATS

MLX Q4
Affine codes + group scales and biases

GGUF Q4
Block scales; K_M mixes precisions

C4W
Signed INT4 + one FP16 scale per row

Runtime / format	bpw†	KLD ↓	PPL ↓	Prefill 1,024 tokens tok/s ↑	Decode 256 steps tok/s ↑
BF16 reference	16.00	0	71.890	—	—
llama.cpp Q4_K_M	4.5+	0.2741	73.295	1,347.4	79.839
Native C4W	**4.005**	**0.3704**	**81.541**	**1,404.3**	**96.289**
llama.cpp Q4_1	5.00	0.4125	89.265	1,474.5	96.552
llama.cpp Q4_0	4.50	0.4497	81.609	1,480.5	85.730
MLX Q4 · FP16	4.50	0.5793	119.308	1,537.5	98.852
Core ML C4W	**4.005**	**0.7344**	**130.329**	**1,057.3**	**11.604**

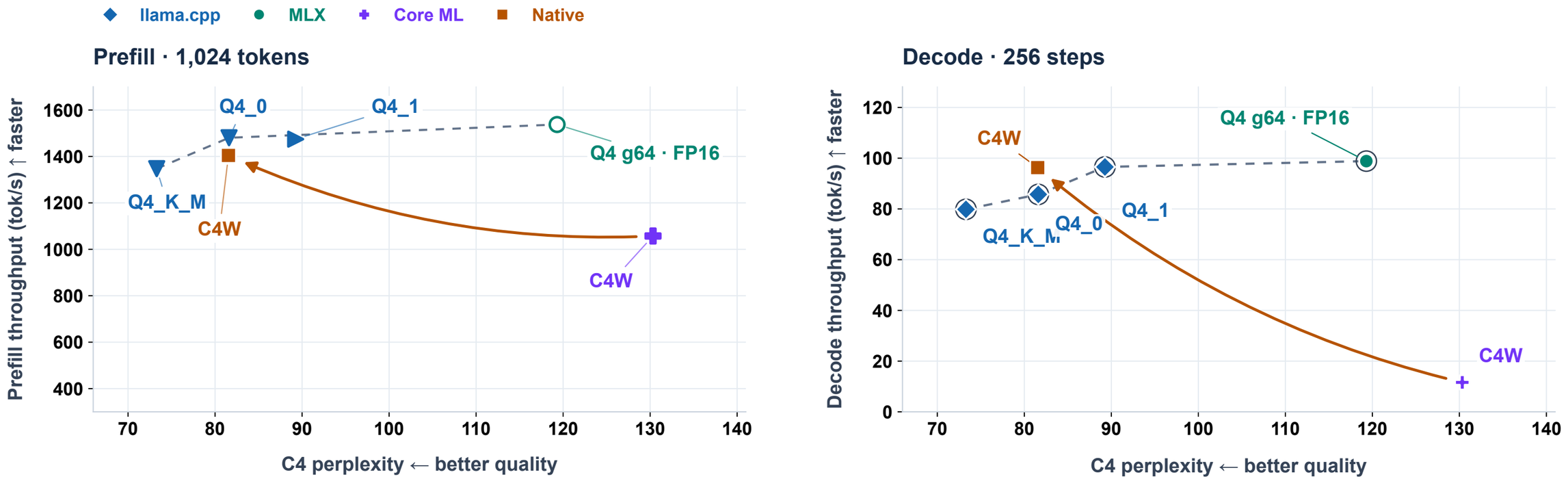
Read the tradeoff

- Lower KLD / PPL means higher quality.
- Lower bpw reduces weight traffic.
- Native C4W uses GPTQ-style calibration on 2,048 C4 tokens to improve quality.
- MLX: FP16 execution; last-token prefill head.

© 2026 Arm

Gemma4-E4B: prefill / decode

APPLE M4 MAX · NON-QAT · C4 QUALITY / APPLE SILICON PERFORMANCE



Implemented autonomously by Loom

- **ExecuTorch export:** built the Core ML text-decoder adapter.
- **Prefill MLP:** fused gate/up + GELU; 10 KiB packed staging + shape specialization.
- **GPTQ-style calibration:** fitted INT4 codes + row scales on 2,048 C4 tokens.
- **Prefill PLE:** tiled projection, 65.1 → 30.5 ms. Combined prefill gain: +5.8%.
- **Prefill attention:** QK/AV blocks; specialized QKV + output projections.
- **Native runtime:** packed INT4, batched dispatch, reused KV/scratch + GPU argmax.

© 2026 Arm

Gemma4-E4B: generated C4W code

GEMMA4-E4B · NATIVE C4W · SELECTED EXECUTION PATHS

Objective-C++ · decoder-layer excerpt · runtime.mm lines 165–190

```
void layer(Command command,uint32_t global,uint32_t count,uint32_t position) {
  bool full=global%6==5;uint32_t local=full?global/6:global-global/6,dim=full?512:256;
  Bank& b=banks[full];
  Gemma4PromptMatrixRmsNormParams norm{local,Hidden,count,1e-6f,0};
  encode(command,Norm,MTLSizeMake(count,1,1),256,norm,{hidden,b.input_norm,normalized},3);
  linear(command,b.q,normalized,qraw,local,count);
  if(global<24){linear(command,b.k,normalized,kraw,local,count);linear(command,b.v,normalized,vraw,local,count);}
  Gemma4PromptMatrixQkNormRopeCacheParams rope{local,8,global<24?2u:0u,dim,8*dim,2*dim,Capacity,
    count,position,full?128u:256u,1u,1e-6f,0};
  uint32_t donor=global<24?global:(full?23:22);
  encode(command,QkRope,MTLSizeMake(global<24?10:8,count,1),256,rope,
    {qraw,kraw,vraw,b.q_norm,b.k_norm,b.cosine,b.sine,query,cache_k[donor],cache_v[donor]},10);
  attend(command,global,local,count,position,dim);
  linear(command,b.o,attention,projected,local,count);
  Gemma4PromptMatrixPostAttentionRmsNormParams post{local,Hidden,count,1e-6f};
  encode(command,PostAttention,MTLSizeMake(count,1,1),256,post,
    {hidden,projected,b.post_attn,b.pre_ffn,residual,normalized},6);
  LoomC4WLinearParams mlp{local,Intermediate,Hidden,count,0,0};
  uint32_t tile=count>=64?64:count>=32?32:16;
  encode(command,count==1?GateUpVector:count<32?SmallGateUp:count<64?MidGateUp:PairedMlp,
    MTLSizeMake(count==1?Intermediate:(count<64?2:1)*((Intermediate+31)/32),count==1?1:(count+tile-1)/tile,1),
    count==1?32:count<32?256:128,mlp,{normalized,b.gate.packed,b.gate.scale,b.up.packed,b.up.scale,
                    count<64?gate:intermediate,up},7);
  if(count<64){Gemma4PromptMatrixMlpIntermediateParams p{Intermediate,count};
    encode(command,Gelu,MTLSizeMake(count*Intermediate,1,1),256,p,{gate,up,intermediate},3,false);}
  linear(command,b.down,intermediate,down,local,count);
```

Host source ↗ Metal kernels ↗ ABI ↗

SOURCE SECTIONS · LINES OF CODE

29 Metal entrypoints

2,074 runtime LoC · 499 Python LoC

Source section	LoC
Objective-C++ host ↗	**309**
Generated Metal Kernels ↗	**1,671**
Host / Metal ABI ↗	**94**
C4W row scales and packing ↗	**139**
GPTQ-style CUDA compensation ↗	**51**
Frozen dense calibration capture ↗	**116**
C4 calibration driver ↗	**94**
Native C4W bank export ↗	**99**
Build, run and scope ↗	**83**

Physical lines, including comments and blanks.
Build, run and scope counts documentation lines.

Download all source ↗

no identified MLX/Core ML code reuse

© 2026 Arm

LOOM · GENERALIZATION

One Loom Harness - Multiple Targets

Applied to Raspberry Pi · Upcoming Phone NX · DGX Spark

arm

© 2026 Arm

Does the Harness generalize?

TRANSFER EVIDENCE · CPU RUNTIME → LITERT NX DELEGATE → VLA INFERENCE

RASPBERRY PI 5 · CPU

YOLO26n

1.87×

faster inference than NCNN

At 12.5 FPS: **47.8% CPU** and **2 cores free**, versus NCNN at 79.2% and 0.

Planned public release: Arm Runtime with Raspberry Pi and Ultralytics.

UPCOMING PHONE

Neural Accelerator with LiteRT

1.39×

geometric-mean speedup

28.27% lower latency across 33 positive controlled model endpoints.

Strategic impact: Graph and dispatch optimizations improve the Neural Accelerator with LiteRT path.

NVIDIA · DGX SPARK (GB10)

GR00T N1.7

1.67×

faster vs initial mixed NVFP4

56.7 → 33.9 ms end-to-end latency on the qualified TensorRT path.

Impact: Lower latency with closed-loop quality checks; task success varies.

The reusable unit is the search and its evidence.

Different implementation layers; the same bounded proposal, independent gates, and durable result record.

© 2026 Arm

YOLO26n CPU runtime on Raspberry Pi 5

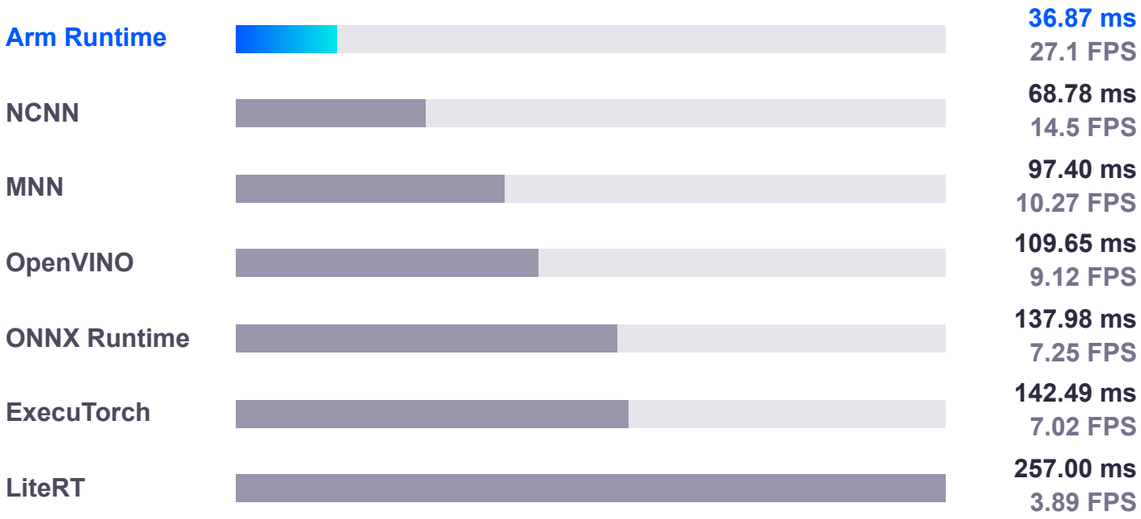
1.87× FASTER—AND TWO CORES FREE AT THE APPLICATION TARGET

● MEASURED

1.87× faster

CPU inference performance

YOLO26n · four CPU threads · lower latency is better



Similar accuracy NCNN **0.4784** Arm Runtime **0.4787**

mAP50–95 · COCO128

Application example

Object-detection target · 80 ms/frame

12.5 FPS

	NCNN	Arm Runtime
CPU usage	**79.2%**	**47.8%**
CPU headroom	**20.8%**	**52.2%**
CPU cores free	**0**	**2**

> Two cores remain available for application logic and connectivity on the same Raspberry Pi 5.

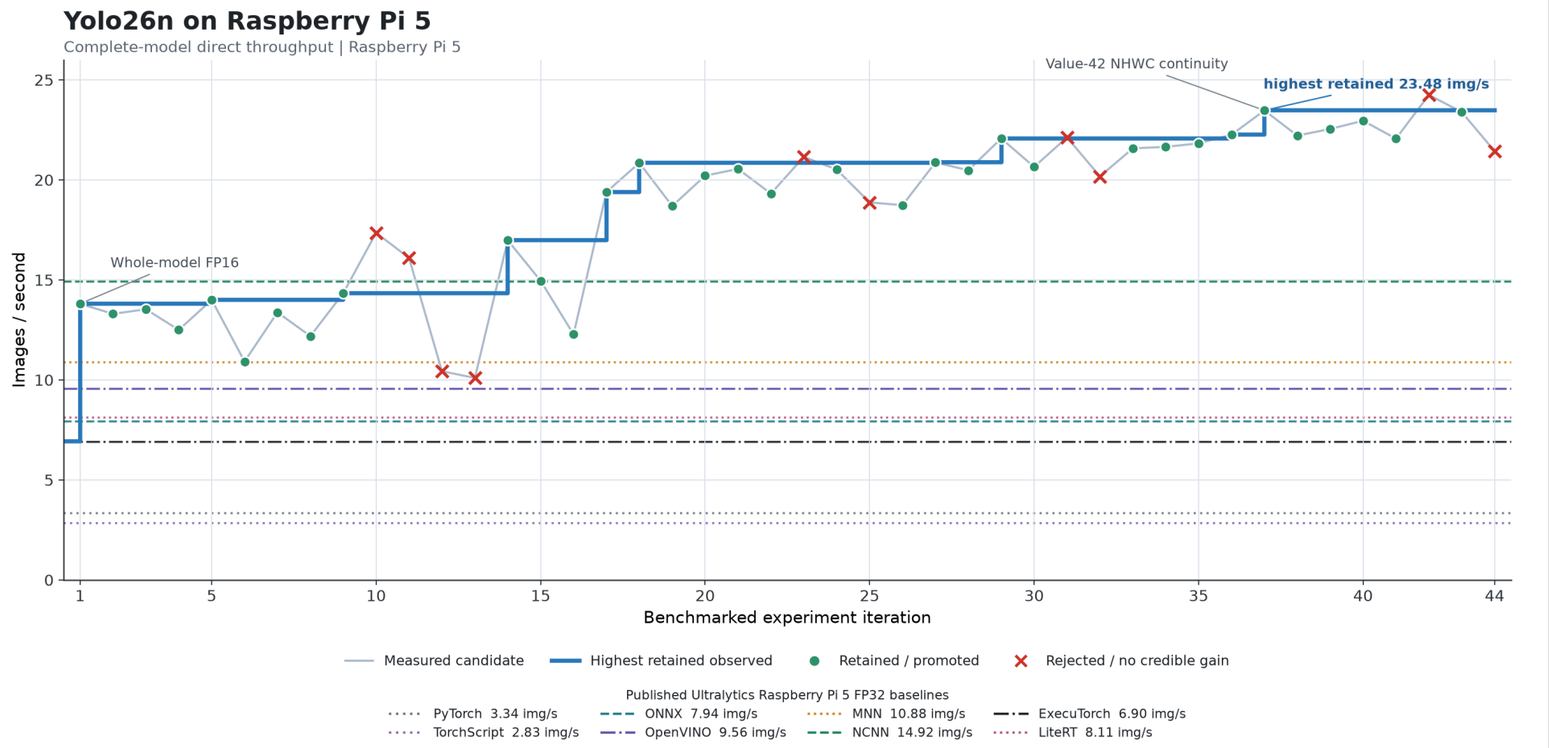
PLANNED PUBLIC RELEASE **Arm Runtime for YOLO26 · joint announcement with Ultralytics and Raspberry Pi**

arm

© 2026 Arm

YOLO26n on Raspberry Pi 5: optimization progress

LOOM: KEEP GAINS. RECORD FAILURES. START FROM THE NEW BEST.



1 · Correctness — valid detections

Run the complete candidate on the same COCO128 images at 640 px. Export and validation must finish with valid boxes, classes, and scores.

2 · Quality — mAP50–95 vs FP32

Candidate: 0.4787 mAP. Published FP32 PyTorch baseline: 0.4760. Treat the result as quality parity, not an accuracy gain.

3 · Performance — images/second

Only after correctness and quality pass, measure complete-model throughput on the same Raspberry Pi 5.

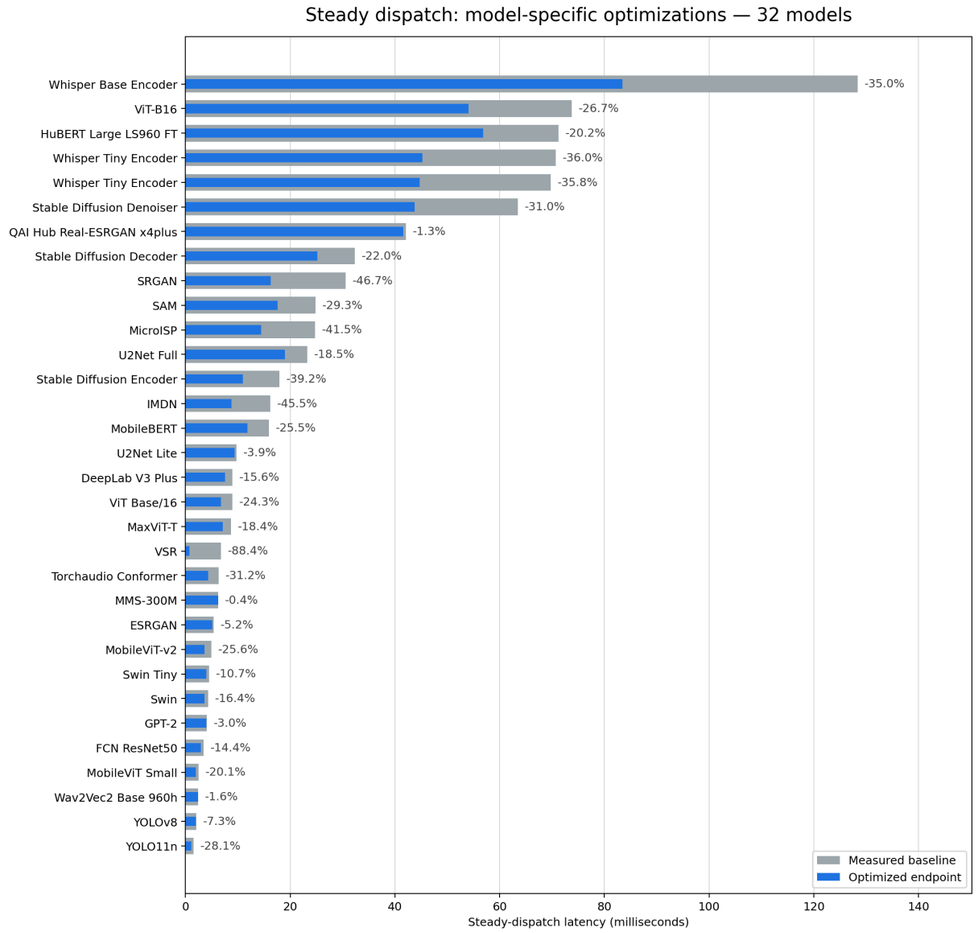
4 · Promotion — advance the ratchet

Only a faster, FP32-quality candidate moves the blue best-so-far staircase.

© 2026 Arm

LiteRT NX delegate on an upcoming phone

33 MODELS IMPROVE ON AN UPCOMING PHONE



28.27%
equal-model geometric-mean dispatch latency reduction

33
accepted model endpoints overall; 32 shown in chart

1.3941×
equal-model geometric-mean dispatch speedup

Selected optimization highlights

VSR · eliminate quantization-dead residuals
6.749 → 0.781 ms · −88.43% dispatch

SRGAN · restructure sub-pixel upsampling
30.566 → 16.288 ms · −46.71% dispatch

IMDN · split distillation convolutions
16.199 → 8.828 ms · −45.50% dispatch

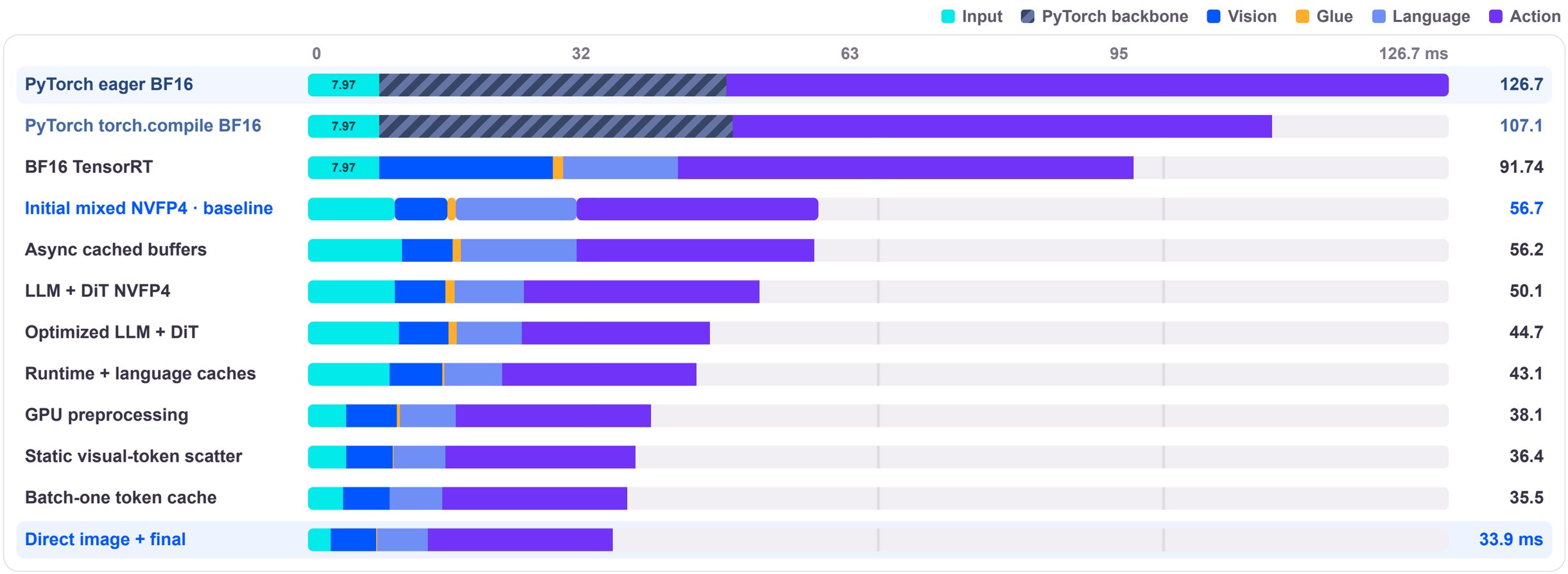
MicroISP · compose exact lookup tables
24.750 → 14.478 ms · −41.50% dispatch

Whisper Tiny Encoder · fold exact softmax operations
70.695 → 45.275 ms · −35.96% dispatch

© 2026 Arm

GR00T N1.7 TensorRT on DGX Spark

1.67× FASTER THAN INITIAL MIXED NVFP4: 56.7 → 33.9 MS



1.67× vs initial mixed NVFP4

40.21% lower end-to-end latency

arm

© 2026 Arm

GR00T on DGX Spark: LIBERO-10 evaluation

TEN MANIPULATION TASKS EXPOSE DIFFERENT POLICY FAILURE MODES

10 tasks × 4 seeds 40 matched episodes per configuration

Same task/seed pairs Every action changes the next observation

Binary task success Goal reward reached before the 720-step cap

LIBERO TASK	BF16	INITIAL NVFP4	LLM + DIT	OPTIMIZED	RUNTIME CACHE	GPU PREPROCESS	STATIC SCATTER	BATCH-ONE	FINAL
Bowl in drawer	4/4	3/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
Mug in microwave	4/4	2/4	4/4	4/4	4/4	4/4	3/4	3/4	3/4
Two moka pots	3/4	3/4	4/4	3/4	3/4	1/4	3/4	1/4	1/4
Stove and moka pot	3/4	4/4	3/4	3/4	3/4	3/4	4/4	4/4	4/4
Soup and cream cheese	4/4	4/4	3/4	4/4	4/4	4/4	3/4	3/4	3/4
Cream cheese and butter	4/4	4/4	3/4	4/4	4/4	4/4	4/4	4/4	4/4
Soup and tomato sauce	4/4	4/4	4/4	3/4	3/4	3/4	4/4	4/4	4/4
Two mugs	3/4	4/4	2/4	3/4	3/4	3/4	3/4	3/4	3/4
Mug and pudding	1/4	4/4	3/4	2/4	2/4	2/4	4/4	4/4	4/4
Book in caddy	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
All tasks	**34/40**	**36/40**	**34/40**	**34/40**	**34/40**	**32/40**	**36/40**	**34/40**	**34/40**

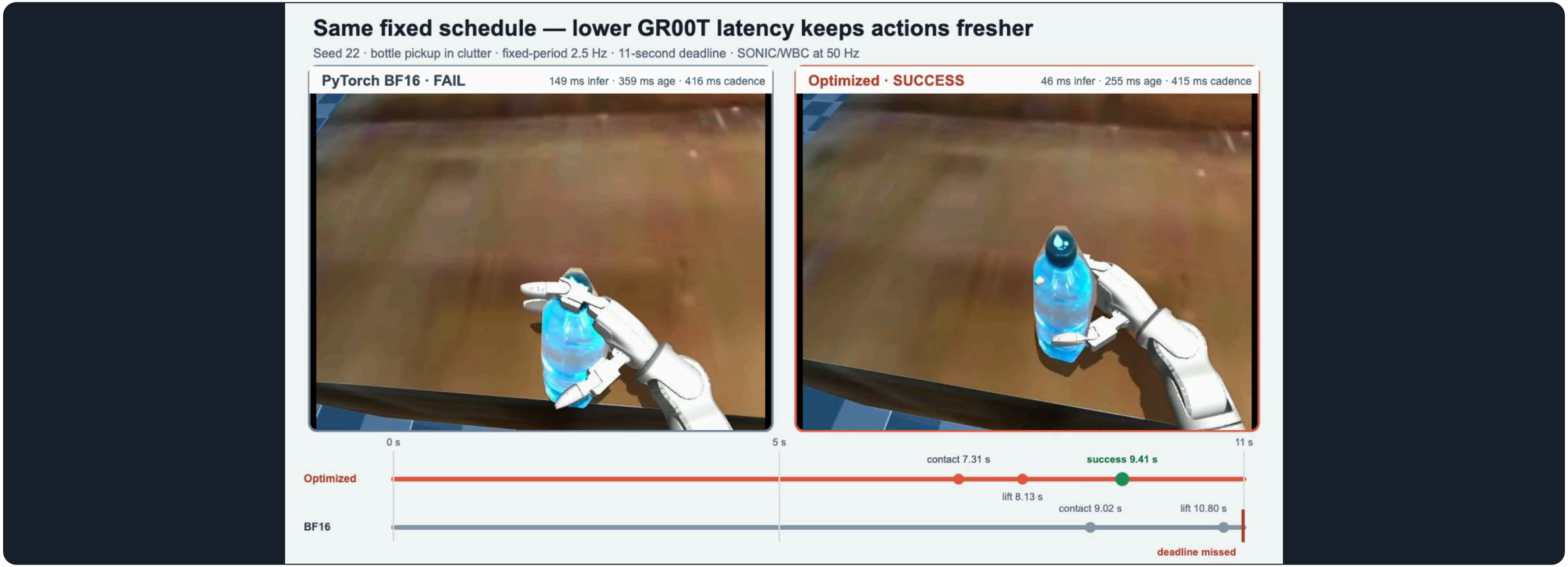
Closed-loop amplification Small numerical action changes alter the next observation, then compound across the rollout.

Not statistically resolved Four seeds per task is a small sample; all 95% success intervals overlap.

© 2026 Arm

GR00T WBC on DGX Spark: action latency

SAME SCHEDULE; FRESHER ACTIONS CHANGE THE OUTCOME



Seed 22 · fixed-period 2.5 Hz · 11-second deadline BF16: 149 ms inference / 359 ms action age / failure · Optimized: 46 ms inference / 255 ms action age / success

No external scene perturbation; camera motion comes from the simulated robot controller.

© 2026 Arm

Conclusions

Quality and performance together.
One measured approach across different targets.

arm

© 2026 Arm

The LOOM harness generalizes

CONCLUSIONS · RESULTS ACROSS MODELS AND HARDWARE

APPLE SILICON · QUALITY AND PERFORMANCE

Qwen3 + Gemma 4 on Apple M4 Max

Agent-generated runtimes can reach the quality–performance Pareto frontier alongside highly optimized general-purpose runtimes: **llama.cpp and MLX**.

YOLO26n · RASPBERRY PI 5

1.87× faster

Generated CPU runtime vs NCNN, with comparable accuracy.

LiteRT · UPCOMING PHONE

1.39× speedup

Geometric mean across 33 models with accepted dispatch gains.

GR00T · DGX SPARK

1.67× faster

TensorRT pipeline: 56.7 → 33.9 ms, with closed-loop quality checks.

Runs autonomously · produces verifiable, reproducible results.

© 2026 Arm

Agents make bespoke runtimes practical

CONCLUSIONS · AGENTS CHANGE THE ECONOMICS

Lower the engineering cost of optimizing each workload × hardware target.

GENERAL-PURPOSE RUNTIMES

Share engineering cost

- Shared operators and dispatch across models and devices.
- Each specialized path adds engineering and maintenance work.

AGENT-DRIVEN SPECIALIZATION

Specialize per workload × target

- Agents build, profile, optimize and verify the real workload.
- Generate runtimes for its model, shapes, precision and hardware.

Workload × target — Fix the model, shapes, quality and resource constraints.

↓

Generate and search — Explore layouts, fusion, kernels, memory and dispatch.

↓

Verify and measure — Check quality and measure the complete workload on target.

↓

Retain and repeat — Keep qualified gains and reuse the evidence for the next target.

Unlock performance through specialization. Generate bespoke runtimes per workload and target, instead of adding every specialized path to a general-purpose runtime.

arm

© 2026 Arm

A paradigm shift in runtime engineering

CONCLUSIONS · MAINTAIN THE HARNESS AND KNOWLEDGE BASE

PyTorch and the open-source ecosystem become the substrate for specialization.

REUSE THE SUBSTRATE

Frameworks, tools and runtimes

PyTorch + open-source frameworks

Model definitions, export and reference behavior

Tools + compilers

TorchAO · profilers · kernel libraries

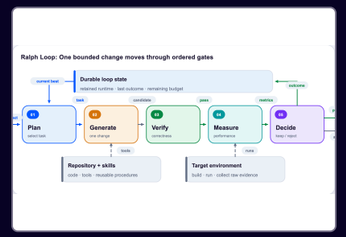
General-purpose runtimes

ExecuTorch · LiteRT · TensorRT
llama.cpp · MLX

MAINTAIN AND COMPOUND

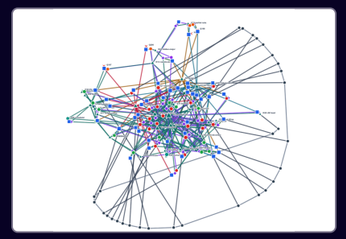
LOOM harness

Generate · profile · optimize · verify
Autonomous workload contracts



Knowledge base

Validated recipes + quality gates.
Versioned code and evidence.



Retain qualified gains → reuse and replay

GENERATE PER WORKLOAD × TARGET

Bespoke, optimized runtimes and kernels

Qwen3 / Gemma4
Apple M4 Max · native Metal

YOLO26n
Raspberry Pi 5 · CPU runtime

LiteRT NX delegate
Upcoming phone · graph optimization

GR00T
DGX Spark · TensorRT pipeline

Scalable, verifiable, reproducible.

- Maintain the harness and knowledge base, instead of extending a general-purpose runtime for every platform.
- Upstream generalizable optimizations to open-source runtimes, tools and kernel libraries.

arm

© 2026 Arm

LOOM · DISCUSSION

Q&A

Questions and discussion

arm

© 2026 Arm