

To our knowledge, the first measured deployment of a masked diffusion LLM on an integrated NPU.

Two graph-surgery fixes unblock a prefill-only, non-autoregressive graph on the vendor Ryzen AI stack — and within its seq ≤ 128 window the NPU becomes the energy-optimal target.

480 ms

NPU INT4 forward at seq=128
±19 ms, n=30 — 1.9× faster than CPU INT4

17.9 J

per forward — 6.3× lower energy
than the CPU at the same INT4

2.7×

lower energy per token than
size-matched 8B AR decode on iGPU

3

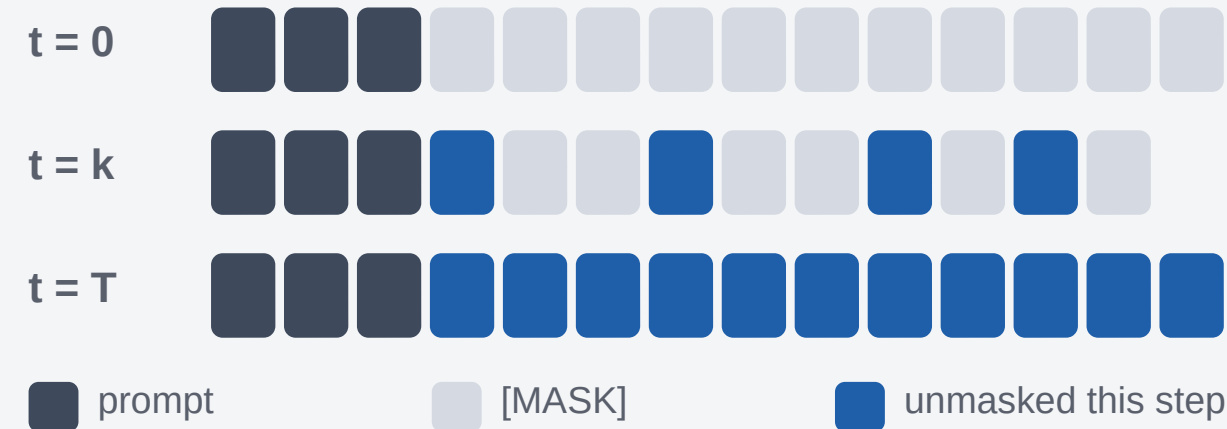
masked-diffusion LLMs deployed
end-to-end, across two lineages

The workload

Why an off-the-shelf NPU runtime refuses this graph.

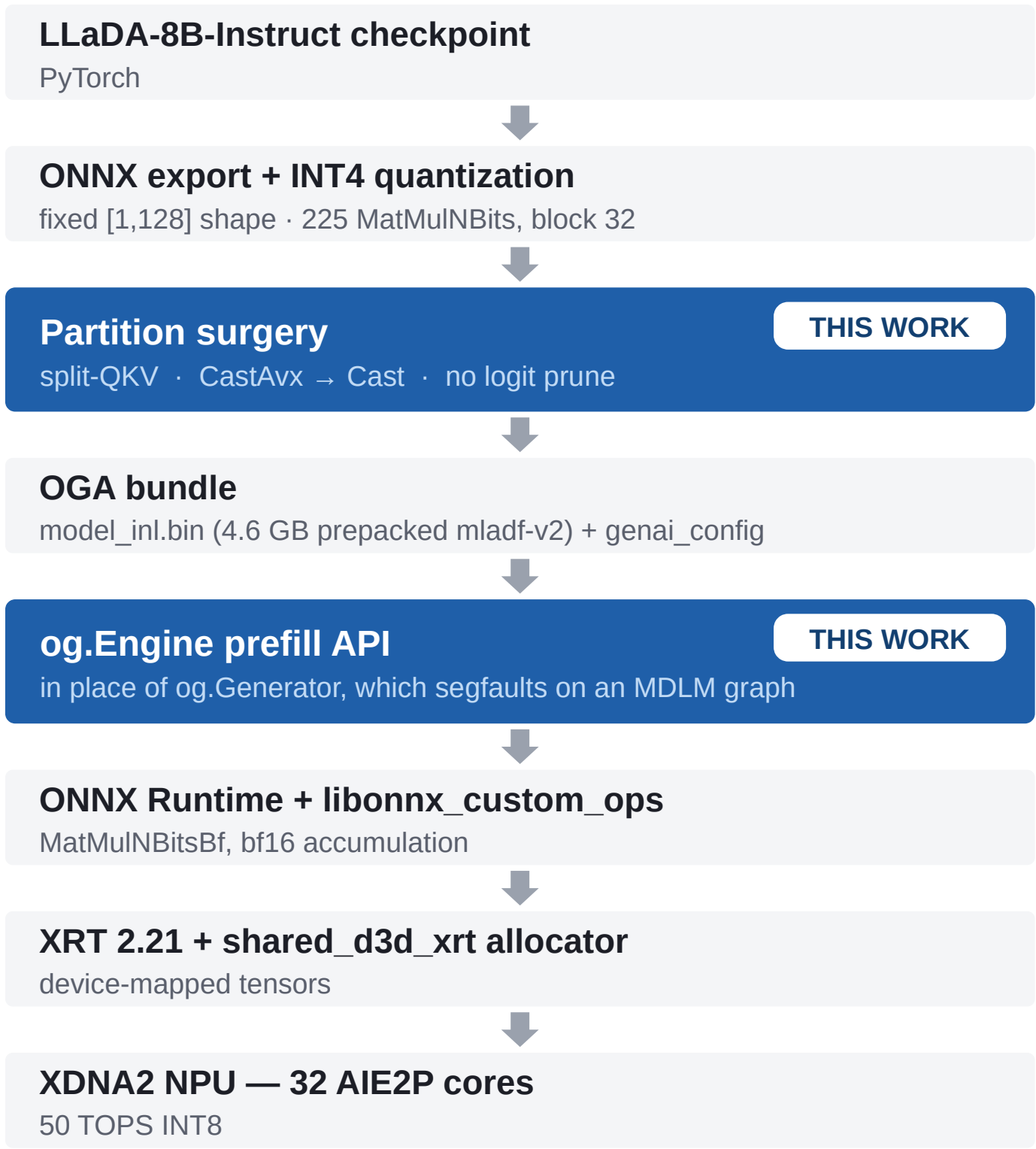
- **One full bidirectional forward per step.** No KV-cache and no left-to-right order — the whole seq=128 window is recomputed at every diffusion step.
- **The vendor runtime assumes autoregressive decode.** onnxruntime-genai 0.9.2 and Ryzen AI 1.6 bake one-token-per-step generation into the API; our graph violates that.
- **Worth solving: the shape fits the silicon.** A diffusion step is one dense, low-precision, dependency-free forward.

Masked-diffusion generation

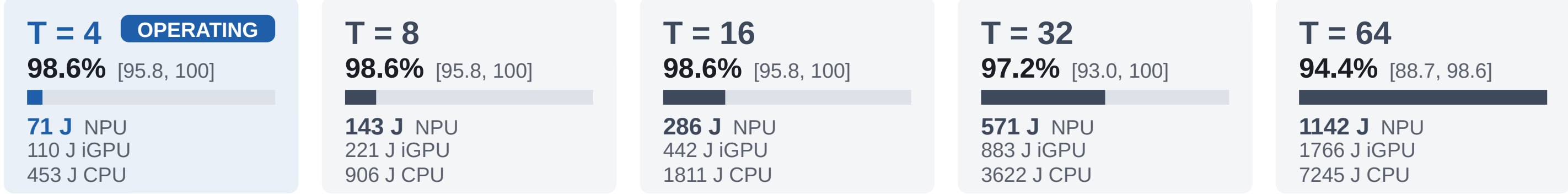


From checkpoint to XDNA2

Grey layers are standard tooling; blue layers are this work.

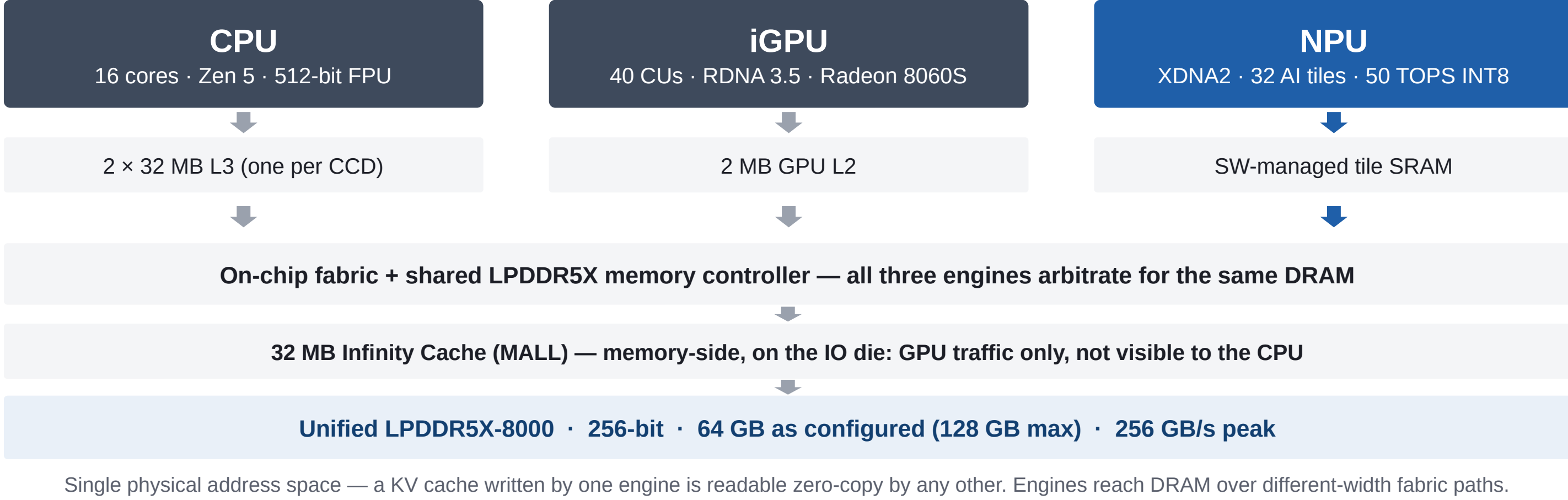


Diffusion steps: the quality / energy frontier



Energy scales linearly with T; quality does not. No accuracy difference across T=4–64 is significant on this near-ceiling eval set.

Platform



Protocol: seq=128, mean ± sd over n=30 after 5 discarded warmups, one process per backend. Latency and energy share one window, so J/forward = W × latency. Ubuntu 24.04, XRT 2.21.75, Ryzen AI 1.6 + 1.7 tooling.

Forward-pass latency

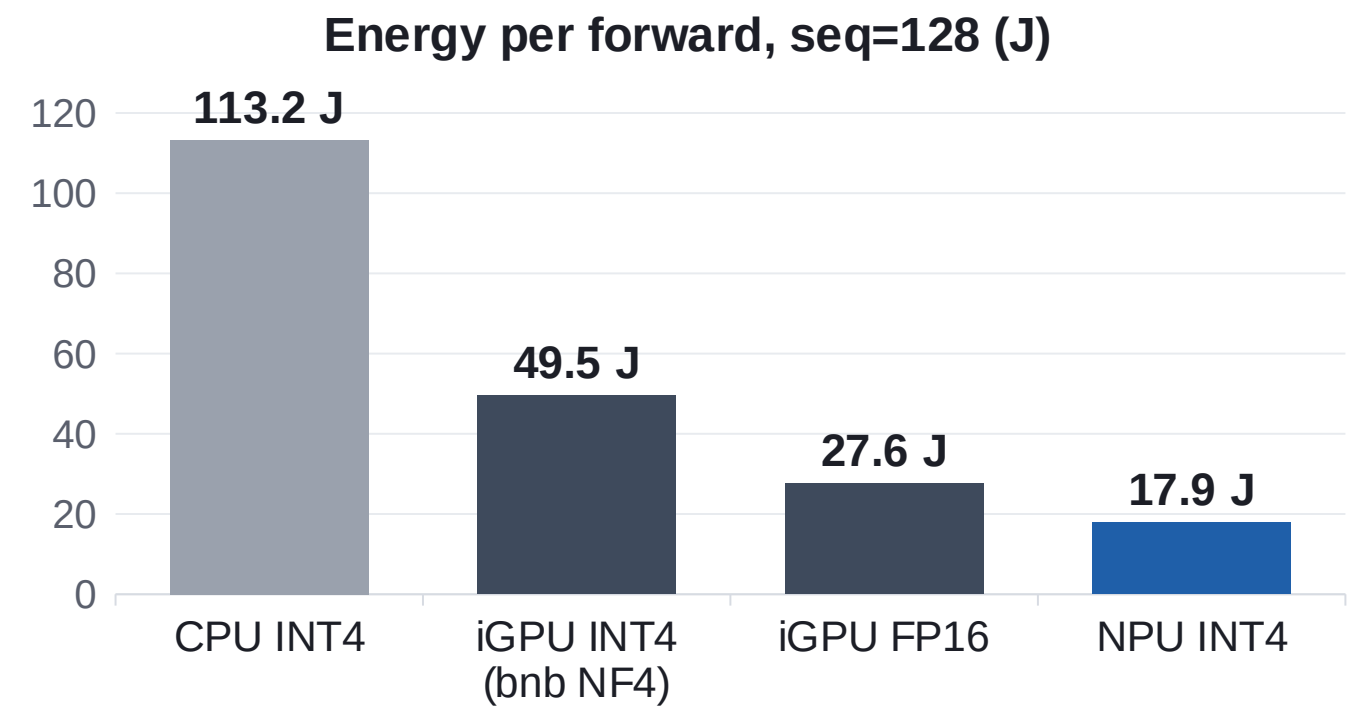
seq=128, mean ± sd, n=30 after 5 warmups.

Path	Device	Latency	vs CPU
CPU INT4 (ORT)	CPU	923 ± 26 ms	1.00×
iGPU INT4 (bnb NF4)	iGPU	644 ms	1.43×
iGPU FP16 (PyTorch)	iGPU	350 ms	2.64×
NPU INT4 (vendor)	NPU	480 ± 19 ms	1.92×
Route C (MLIR-AIE)	NPU	8700 ms	0.11×

At identical INT4 quantization the NPU is 1.9× faster than the CPU, with lower variance. The iGPU wins absolute latency in FP16 — but latency alone does not pick the target.

Energy is the real story

Package RAPL × measured latency, from one consistent window.

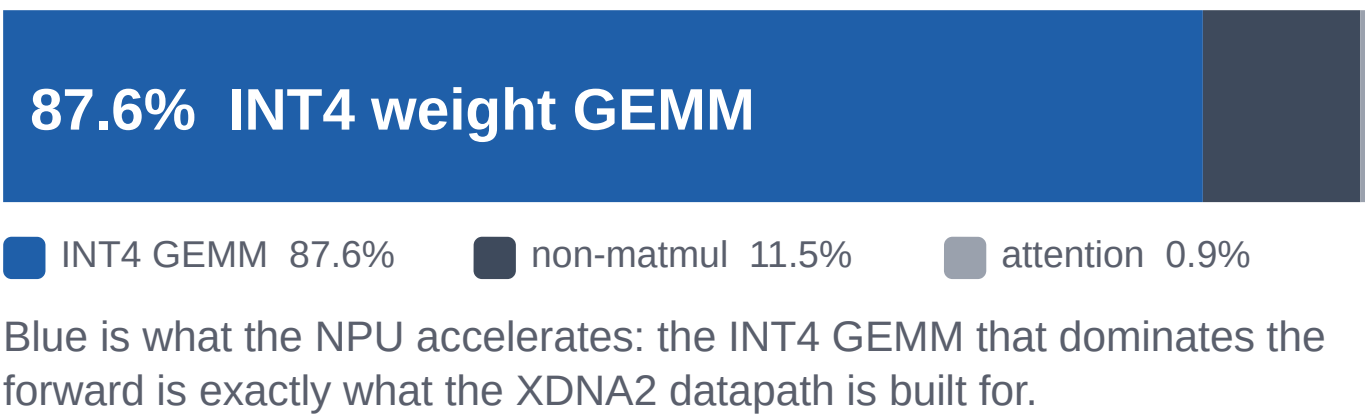


36.7 W on the NPU — about 10 W above the 27 W idle baseline — against 120 W for the CPU and 83 W for the iGPU: 1.55× less energy per forward than the best iGPU configuration we measured.

Caveat: XDNA2 exposes no isolable power rail, so every NPU joule here is whole-package RAPL including driver and dispatch cost — 6.3× is a floor.

Where the time goes

ORT node-level profile of one steady-state CPU forward: 1833 nodes, 899 ms.



Window: the NPU path is capped at seq=128 — at 256 the first quantized matmul faults with “M size exceeding ultimate shape limit”, and longer contexts fall back to the iGPU.

INT4 is near-lossless

vs an FP32 PyTorch reference on 32 passages. Perplexity here is masked-token pseudo-PPL, not autoregressive PPL.

Metric	INT4 vs full precision
Logit cosine similarity	0.996
Top-1 argmax agreement	99.1%
KL(FP II INT4)	0.030 nats
Masked-token acc. (FP / INT4)	86.9% / 87.6%
Pseudo-perplexity (FP / INT4)	1.798 / 1.724

Corpus-independent anchor: 75.2% ± 1.1% masked-token accuracy on held-out WikiText-2 (10,440 evaluations). Generation quality is scored by a curated short-answer proxy, not a full task suite.

The recipe transfers

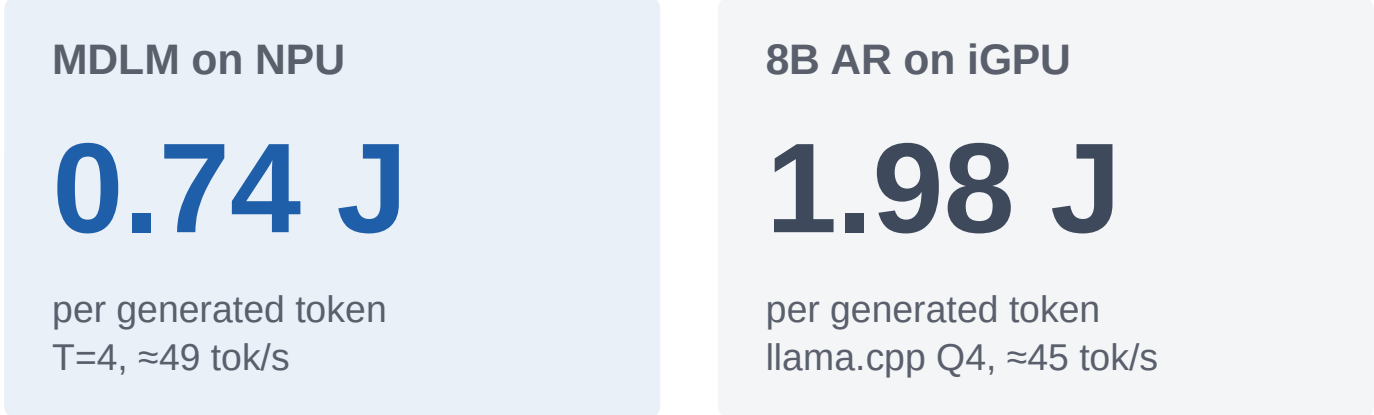
Same export → quantize → split-QKV → CastAvx → og.Engine pipeline, three models, two lineages.

Model	NPU forward	J/fwd	vs its CPU
LLaDA-8B-Instruct native diffusion	480 ± 19 ms	17.9	1.92× / 6.3×
DiffuLLaMA-7B AR-adapted LLaMA-2	555.8 ± 21.5 ms	21.4	1.67× / 5.2×
LLaDA-1.5 RL / preference-tuned	550.7 ± 33.8 ms	21.0	1.85× / 5.8×

DiffuLLaMA is an autoregressive LLaMA-2 adapted to diffusion — a different lineage entirely — and needed only two mechanical adaptations: a bidirectional export patch and a 2D → 3D layout collapse.

Versus autoregressive decode

Energy per generated token — the framing-independent cost of output.



- **Parallel unmasking:** 24 tokens resolved per forward at Lg=96, T=4 — 24× fewer forwards than AR.
- **Step reduction:** four steps already suffice on this eval set (98.6%, CI [95.8, 100]); no difference across T=4–64 is significant.
- **Hardware:** each forward costs 6.3× less energy on the NPU. The levers multiply.

What that buys on a laptop

- **Battery.** A 72 Wh battery affords ≈3.6k short (T=4, 96-token) generations on the NPU versus ≈0.6k on the CPU — 71 J against 453 J each.
- **Concurrency.** ~10 W above idle on otherwise-idle silicon leaves the CPU and iGPU budgets free for the foreground app.
- **Private infilling.** Bidirectional generation suits fill-in-the-middle work — docstrings, form fields, short rewrites — entirely on-device, inside the seq ≤ 128 window.

Code, ONNX bundles and repro scripts
github.com/raviguptaamd/llada-npu-inference

Built on AMD Ryzen AI SDK 1.6 / 1.7 · Vitis AI Execution Provider · MatMulNBitsBf and libonnx_custom_ops · ryzenai_onnx_utils · XRT 2.21 + amdxdna · IRON / MLIR-AIE and Peano. Graphs run on ONNX Runtime and onnxruntime-genai.