Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Per-Node Performance and Memory Limits of LLM Inference on CPU-Only Edge Nodes

Yogesh Narayan Gaur

BL.EN.R4CSE22014@bl.students.amrita.edu

Amrita School of Computing, Bengaluru, Amrita Vishwa Vidyapeetham

Manju Khanna

Amrita School of Computing, Bengaluru, Amrita Vishwa Vidyapeetham

Research Article

Keywords: Edge computing, CPU inference, Large language models, Performance characterization, Quantization

Posted Date: October 20th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11101926/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Per-Node Performance and Memory Limits of LLM Inference on CPU-Only Edge Nodes

Yogesh Narayan Gaur[1*] and Manju Khanna[1]

[1*]Department of Computer Science and Engineering, Amrita School of Computing, Bengaluru, Amrita Vishwa Vidyapeetham, Bengaluru, India.

*Corresponding author(s). E-mail(s): BL.EN.R4CSE22014@bl.students.amrita.edu;
Contributing authors: k_manju@blr.amrita.edu;

Abstract

Distributed edge deployments increasingly place language model inference on nodes with no GPU, serving requests from a general-purpose CPU. Planning such a deployment requires knowing what one node can deliver, yet CPU-only inference remains poorly characterized. This paper measures three models of 1.1–7 billion parameters at three precision levels on a representative edge node, with five replicates per configuration. One bottleneck explains the results: the node is limited by memory bandwidth, not compute. Utilization reaches 97% while a 7B model sustains 3.76 tokens per second, so a node saturates on one request and concurrency must come from further nodes. Runtime memory also exceeds weight size, and quantization raises the total by up to 48% rather than reducing it, opposite to GPU behaviour. A calibrated sizing model predicts per-node memory from parameter count and precision within 2.51% mean error: a 7B node needs 20 GB, not 14 GB.

Keywords: Edge computing, CPU inference, Large language models, Performance characterization, Quantization

1 Introduction

Large language models (LLMs) have transformed natural language processing, achieving strong results across tasks from conversational assistants to code generation.

Modern models contain billions of parameters, from roughly one billion in compact variants to seventy billion in state-of-the-art systems. Each parameter requires two to four bytes of storage, so a seven-billion-parameter model needs approximately 14 GB in half precision.

1.1 Inference at the edge tier

Distributed edge architectures push computation away from centralized datacentres and towards nodes near the data source. A deployment is typically organized as a tier of many modest nodes, each serving local requests, with a cloud tier held in reserve for work the edge cannot absorb. Placing language model inference in this tier is attractive for privacy-sensitive data, for sites with unreliable connectivity, for latency-critical control loops, and for operators who wish to avoid recurring per-request cloud charges.

Planning such a deployment requires a per-node answer to three questions: how many concurrent requests a single node can sustain, how much memory each node must be specified with, and which requests must be offloaded to the cloud tier because the local node cannot serve them within the latency budget. Each answer depends on measured per-node behaviour. Aggregate cluster throughput is the product of node count and per-node capability, so an error in the per-node figure propagates through the entire capacity plan. Schedulers that place work across such tiers optimize against per-node cost models [1, 2], and the quality of any placement decision is bounded by the accuracy of the per-node figures supplied to it.

Most edge nodes lack a discrete GPU and rely on general-purpose CPUs, chosen for cost, thermal envelope, and form factor. Computation is commonly offloaded to an accelerator when one is present [3], but that option is unavailable at this tier. Understanding how these models behave on CPU-only nodes is therefore a prerequisite for edge capacity planning.

1.2 The knowledge gap

Running neural networks on constrained edge hardware is an established research area. Recurrent architectures in particular have been surveyed for edge intelligence [4] and evaluated systematically across edge deployment scenarios [5], establishing both the metrics and the methodology such studies require.

Language models have not received the same treatment on CPU-only hardware. GPU-based inference has been studied extensively, with quantization reported to reduce memory by 50–75% and to deliver 1.5–3× speedups. CPU-based inference receives far less attention and is usually mentioned only as a fallback in GPU-focused work. The systematic per-node evaluation that exists for earlier model families is therefore missing for the models now being deployed, which leaves the questions above unanswered for anyone specifying edge hardware:

- What limits a CPU-only node, computation or memory?
- How is memory actually consumed during inference on such a node?
- Do GPU optimization techniques transfer to these nodes?
- What throughput can a single node sustain?

Without measurements, operators size edge tiers by extrapolating from GPU benchmarks, which produces both over-provisioned hardware budgets and nodes that cannot meet their latency targets.

1.3 Research approach

This paper characterizes language model inference on a CPU-only edge node through direct measurement. The aim is not to propose a new optimization but to establish what the platform does.

Quantization serves as a diagnostic instrument. Testing three precision levels (FP16, INT8, INT4) across three model sizes (1.1B, 2.7B, 7B parameters) varies the computational and memory demand placed on the node. Because quantization behaves predictably on GPUs, any deviation measured on a CPU exposes platform-specific behaviour.

1.4 Contributions

- *Platform dependency:* quantization on these nodes increases memory by 40–48% rather than reducing it, opposite to GPU behaviour. The reversal is consistent across a 6× parameter range with CV $< 2\%$.
- *Bottleneck identification:* CPU-only inference is memory-bandwidth limited, not compute limited, across standard precision levels. Utilization reaches 93–99% while throughput remains between 0.06 and 9.79 tokens per second.
- *Memory profiling:* measured runtime memory exceeds nominal weight size by a substantial and precision-dependent margin, so a 7B model requires about 20 GB per node rather than the 14 GB implied by weight size.
- *Per-node memory sizing model:* a calibrated expression that predicts runtime memory from parameter count and precision, validated against all nine configurations with 2.51% mean absolute error. It reduces in practice to multiplying nominal weight size by 1, 3 or 6 for FP16, INT8 and INT4.
- *Per-node capacity reference:* measured throughput and memory figures usable directly as inputs to edge-tier sizing and offload decisions.

2 Related work

2.1 Neural network quantization

Quantization reduces numerical precision to compress models and accelerate inference. Early work addressed 8-bit quantization for convolutional networks [6]. The LLM.int8() method [7] introduced mixed-precision decomposition, enabling quantization of models up to 175 billion parameters with negligible quality loss on GPU platforms.

More aggressive 4-bit quantization followed as models grew. GPTQ [8] applies optimal brain quantization to generative models, reaching 3–4 bits per weight while maintaining accuracy. These techniques report strong GPU results but rarely report CPU performance.

The bitsandbytes library implements CUDA kernels for GPU quantization and is widely adopted for memory-constrained deployment. Its documentation mentions a CPU fallback but provides no performance characterization. This work measures that fallback path systematically.

2.2 Edge-optimized inference

Recognizing that edge hardware differs from datacentre hardware, several CPU-oriented inference frameworks have been developed. GGML, now GGUF [9], provides custom quantization formats with Single Instruction, Multiple Data (SIMD) optimization for CPU architectures. ONNX Runtime [10] includes CPU-specific quantization. TensorFlow Lite [11] targets mobile deployment with optimized INT8 kernels.

These efforts demonstrate that CPU-specific optimization is both necessary and effective. They focus on implementation rather than characterization. This work complements them by explaining why CPU-specific handling is required.

2.3 Serving systems and edge tiers

Several studies analyse how these models run in production. Orca [12] examined distributed serving and documented the split between prefill and decode phases. FlexGen [13] examined memory offloading strategies.

Both target GPU-based datacentre hardware, where a scheduler distributes work across accelerators. An edge tier inverts several of these assumptions: nodes are numerous but individually weak, each typically serves a single local request rather than a batched queue, and no accelerator is available to offload to. Per-node characterization is therefore the appropriate unit of analysis, and it is what this work provides.

3 Methodology

3.1 Node configuration

Experiments are conducted on a configuration representative of an edge node. The processor is an Intel Core Ultra 7 258V, representing modern high-end laptop and edge server CPUs, with 32 GB of DDR5-5600 memory, sufficient for models up to seven billion parameters. Models are loaded from an NVMe SSD, and the operating system is Linux 6.8.0 on Ubuntu.

No discrete GPU is present, and all inference runs on the CPU. This matches edge deployments where accelerators are excluded by cost, power, or form factor.

This study deliberately characterizes commodity edge CPUs on which the available quantization software does not exploit low-precision instructions. Server-class processors with hardware acceleration for low-precision arithmetic (Intel Advanced Matrix Extensions, AMX, or AVX-512 Vector Neural Network Instructions, VNNI) fall outside the scope of this edge-focused study and are expected to behave differently; their characterization is left to future work.

3.2 Experimental configuration

Three models are tested using PyTorch-based quantization at three precision levels. The software stack is PyTorch 2.1.0 (CPU-only), Transformers 4.36.2 for model loading, and bitsandbytes 0.41.3 for quantization. Bitsandbytes is used specifically because it is widely adopted for GPU quantization, which permits direct comparison between intended GPU behaviour and actual CPU behaviour.

The three models span one order of magnitude in parameter count: TinyLlama-1.1B-Chat-v1.0 (1.1B parameters), Phi-2 (2.7B parameters), and Mistral-7B-Instruct-v0.2 (7B parameters). Each is evaluated at three precision levels: FP16, using 16 bits and 2 bytes per parameter, as the baseline; INT8, using 8 bits and 1 byte per parameter, with an expected 50% memory reduction on GPU; and INT4, using 4-bit NormalFloat (NF4) and 0.5 bytes per parameter, with an expected 75% memory reduction on GPU. This yields nine configurations, each run with five replicates.

3.3 Measurement protocol

The task is the prompt "Explain what is machine learning in simple terms" (11 input tokens), generating approximately 100 output tokens. Recorded metrics are total generation time, throughput in tokens per second, memory as peak and average resident set size, and processor utilization as average and peak. Per-token latency is not instrumented, so no prefill/decode split is reported; see Section 5.4. A background thread samples every 100 ms using psutil. Each configuration runs five times in randomized order with cooldown periods to prevent thermal throttling. Results are reported as mean with 95% confidence intervals, and the coefficient of variation ($\text{CV} = \sigma/\mu$) assesses reproducibility.

This yields 45 runs in total. Throughput is computed as generated tokens divided by measured wall-clock generation time, and memory as the peak resident set size of the inference process over the run.

For memory accounting, only the process total is measured. Components are not separately instrumented; the decomposition in Table 3 is an estimate constructed from architecture specifications and is reported for interpretation rather than as measurement. See Section 4.2.

4 Results and analysis

4.1 The node bottleneck

Throughput and utilization are measured together, with good reproducibility (Mistral-7B throughput CV $< 9\%$, memory CV $< 2\%$). Table 1 reports both for every configuration of the two larger models. Across the six configurations of the 2.7B and 7B models, processor utilization reaches 93–99% while sustained throughput spans 0.06 to 9.79 tokens per second, a range of two orders of magnitude. High utilization combined with low throughput indicates a memory-bandwidth limit rather than a compute limit for FP16 and INT8, and computational inefficiency in the emulated arithmetic path for INT4. Throughput is reported for these two models because their replicates ran

to the full output length; the 1.1B model terminated early in several replicates and is therefore excluded from throughput comparison, as noted in Section 5.4.

Table 1 Processor utilization and throughput ($n = 5$ per row)

Model	Precision	CPU util. (%)	Throughput (tok/s)	Pattern
Phi-2	FP16	93.2	9.79 ± 1.00	Memory-bound
Phi-2	INT8	96.9	4.19 ± 0.36	Memory-bound
Phi-2	INT4	99.1	0.17 ± 0.00	Compute-bound
Mistral-7B	FP16	96.9	3.76 ± 0.32	Memory-bound
Mistral-7B	INT8	99.1	1.22 ± 0.01	Memory-bound
Mistral-7B	INT4	97.6	0.06 ± 0.00	Compute-bound

For deployment, the practical consequence is that a node is saturated by a single request. Utilization is already near 100% while serving one stream, so additional concurrent requests on the same node cannot be absorbed; concurrency must come from additional nodes rather than from additional threads.

4.2 Memory profiling

Memory scales with parameter count (Table 2a).

Table 2 Memory consumption analysis. Panel (a) reports the FP16 baseline; panel (b) reports the change under quantization

(a) Memory usage versus model size (FP16 baseline)

Model	Parameters	Peak RAM (GB)	GB per billion
TinyLlama	1.1B	2.44 ± 0.00	2.22
Phi-2	2.7B	5.35 ± 0.01	1.98
Mistral-7B	7.0B	13.77 ± 0.21	1.97

(b) Memory change with quantization (GB)

Model	FP16	INT8	ΔINT8	INT4	ΔINT4
TinyLlama	2.44	3.49	+43%	3.44	+41%
Phi-2	5.35	7.75	+45%	7.75	+45%
Mistral-7B	13.77	20.45	+48%	17.69	+28%

Theory predicts $2N$ gigabytes for N billion parameters at FP16. Measurements track theory within 2–10%, confirming that weights dominate at FP16.

Quantization then produces an unexpected result. On GPUs, memory falls in proportion to bit width, because GPUs compute directly on low-precision values. The GPU expectation is that INT8 reduces memory by 50% and INT4 by 75%; the measured CPU node reality is that INT8 increases memory by 40–48% and INT4 by 28–45%.

The reversal is consistent across all three models with CV $< 2\%$, indicating systematic behaviour. Figure 1 shows this platform-dependent difference.

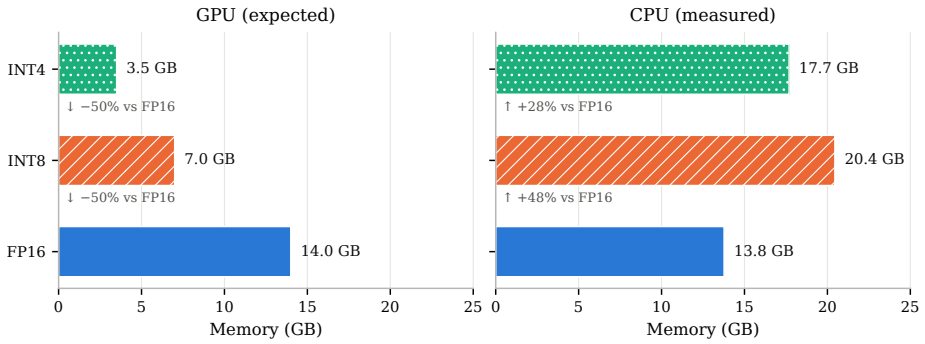


Fig. 1 Memory paradox: GPU quantization reduces memory by 50% while CPU quantization increases it by 48% through dual-representation overhead

The mechanism is dual representation. On the CPU node, quantization stores weights in compressed form, but the tested path cannot compute on them and must convert to FP16 for every operation. Compressed weights are therefore held alongside FP16 conversion buffers, while activations and the attention cache remain in FP16 throughout. Weight compression is outweighed by growth elsewhere.

Table 3 gives an estimated account of where the memory goes, and Figure 2 shows the same estimate graphically. Only the total is measured; the individual components are not separately instrumented. Weight storage is computed from parameter count and precision, activations and KV-cache are estimated from the published architecture specifications of each model [13], and the residual is attributed to conversion buffers, quantization metadata, and framework structures. The decomposition is therefore an interpretation consistent with the measured total, not an independent measurement of each term, and it is offered to explain the mechanism rather than to establish it. The evidence for the mechanism is the measured total itself, which rises by 40–48% under quantization where weight compression alone predicts a fall.

4.3 A per-node memory sizing model

Measured runtime memory exceeds nominal weight size by a margin that varies systematically with precision, which invites a predictive model. The model is built on the measured totals of Table 2 alone and does not depend on the estimated decomposition of Table 3. Runtime memory is divided into weight storage, which is known in advance from parameter count and precision, and a remainder covering everything else the process holds. Weight storage for a model of P billion parameters at a precision of b_q bytes per parameter is

$$M_w(P, q) = P\, b_q. \tag{1}$$

Table 3 Estimated memory decomposition (Mistral-7B, GB). Components are estimated as described in the text; only the total row is measured

Component	FP16	INT8	INT4	Basis
Model weights	13.0	7.0	3.5	Computed from parameter count
Activations	0.5	4.0	4.0	Estimated from architecture
KV-cache	0.2	4.0	4.0	Estimated from architecture
Dequant buffers	0.0	3.5	4.2	Residual attribution
Scale factors	0.0	0.5	0.5	Residual attribution
Framework	0.07	1.45	1.49	Residual attribution
Total (measured)	13.77	20.45	17.69	psutil peak RSS, $n = 5$, CV $< 2\%$

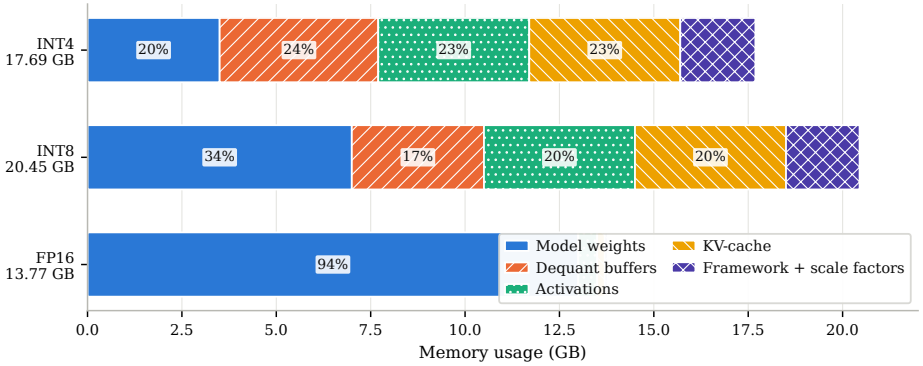


Fig. 2 Estimated memory decomposition for Mistral-7B, plotting the Table 3 values with scale factors folded into the framework segment. Quantization shrinks weights but adds buffers and overhead, raising the total. Only the total is measured

The remainder is modelled with the smallest functional form that the data support: one term proportional to parameter count and one independent of it,

$$M_o(P, q) = \alpha_q P + c_q, \tag{2}$$

where α_q is the per-parameter overhead of precision q in gigabytes per billion parameters and c_q is a constant in gigabytes. The linear form is not assumed a priori. It is adopted because the residuals $M_{\text{meas}} - P b_q$ are linear in P at every precision, as the parity plot of Figure 3 confirms: all nine points fall within a 6% band with no visible curvature. The physical reading—a per-parameter term for structures that scale with model dimension and a fixed term for framework structures that do not—is consistent with the estimate in Table 3, but the model is calibrated and validated against measured totals, so it stands independently of that estimate. Total runtime memory on a node is the sum of (1) and (2),

$$M(P, q) = P\, b_q + \alpha_q P + c_q, \tag{3}$$

which may be written in the equivalent form

$$M(P, q) = (b_q + \alpha_q)P + c_q, \tag{4}$$

making explicit that memory is linear in parameter count at fixed precision, with a slope that quantization changes. Here b_q is known a priori (2.0 for FP16, 1.0 for INT8, 0.5 for INT4), while α_q and c_q are calibrated per precision by least squares on the residual $M_{\text{meas}} - Pb_q$ over the three measured model sizes. Table 4 lists the resulting coefficients.

Table 4 Calibrated coefficients (from 45 runs)

Precision	b_q (GB/B)	α_q (GB/B)	c_q (GB)
FP16	2.0	−0.072	0.249
INT8	1.0	1.891	0.158
INT4	0.5	1.893	1.012

Two features of these coefficients are notable. First, α for FP16 is effectively zero, confirming that weights account for essentially all memory at full precision. Second, the two quantized precisions yield almost identical overhead coefficients, 1.891 and 1.893 GB per billion parameters. This equality was not imposed by the fit. It indicates that the dual-representation penalty is a property of the quantized execution path itself and does not depend on how aggressively weights are compressed: halving the weight bits again does not halve the buffers, activations, and cache that surround them.

Table 5 evaluates the calibrated model against every measured configuration.

Table 5 Model validation against all nine measured configurations

Model	Precision	P	Measured (GB)	Predicted (GB)	Error (%)
TinyLlama	FP16	1.1	2.443	2.370	−3.0
TinyLlama	INT8	1.1	3.494	3.338	−4.5
TinyLlama	INT4	1.1	3.440	3.644	+5.9
Phi-2	FP16	2.7	5.353	5.454	+1.9
Phi-2	INT8	2.7	7.750	7.964	+2.8
Phi-2	INT4	2.7	7.752	7.473	−3.6
Mistral-7B	FP16	7.0	13.770	13.743	−0.2
Mistral-7B	INT8	7.0	20.454	20.395	−0.3
Mistral-7B	INT4	7.0	17.686	17.762	+0.4

Mean absolute percentage error is 2.51% and the largest single error is 5.9%, occurring for the smallest model where the constant term dominates. Accuracy improves with model size, which is the regime that matters for node specification.

Figure 3 plots predicted against measured memory for all nine configurations. Every point falls within the 6% band, and no systematic curvature is visible, which supports the linear form adopted in (3).

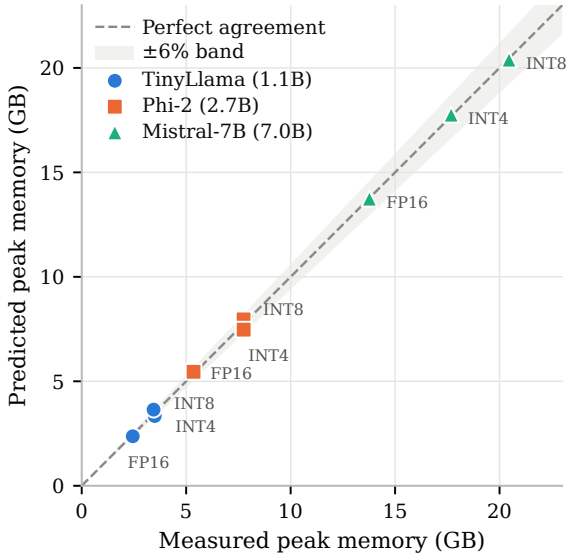


Fig. 3 Predicted versus measured peak memory for all nine configurations. The dashed line marks perfect agreement and the shaded band 6%. All points fall inside the band, and accuracy improves with model size

For deployment use the model reduces to a multiplier on weight size. Dividing (3) by (1) and neglecting the constant term gives

$$k_q = \frac{M(P, q)}{P\, b_q} = 1 + \frac{\alpha_q}{b_q}, \tag{5}$$

so the multiplier depends only on the ratio of overhead to weight bytes. Evaluating (5) against the measured data gives $k = 1.03$ for FP16, 2.99 for INT8 and 5.68 for INT4. An operator sizing a node can therefore multiply nominal weight size by roughly 1, 3 or 6 according to precision. Equation (5) also explains why the multiplier grows as precision falls even though α_q is almost unchanged between INT8 and INT4: the overhead stays fixed while b_q halves, so the ratio doubles. The FP16 multiplier near unity and the INT8 multiplier near three quantify the penalty directly, in that an INT8 deployment consumes about three times its nominal weight size.

The coefficients are calibrated on three model sizes per precision, so the model is presented as an empirically calibrated sizing rule validated across a 6× parameter range rather than as a general law. Its value for planning is that it converts a parameter count and a precision into a memory figure without requiring the operator to run the model first, and it is accurate to within 6% across every configuration tested.

4.4 Performance patterns

Quantization degrades generation speed substantially (Table 6). At the FP16 baseline, Phi-2 reaches nearly 10 tokens per second (100 ms per token), which supports interactive use, while Mistral-7B reaches 3.76 tokens per second (266 ms per token), slower but usable. At INT8, throughput falls 57–68%, with Mistral-7B dropping to 1.22 tokens per second (818 ms per token). At INT4, throughput collapses by 98%: Mistral-7B yields 0.06 tokens per second (16.7 s per token), which is unusable.

Despite the degradation, quantized performance is highly reproducible (CV 0.8–2.8%), indicating systematic overhead rather than random variation.

Table 6 Throughput comparison (tokens per second)

Model	FP16	INT8	INT8 loss
Phi-2	9.79 ± 1.00	4.19 ± 0.36	−57%
Mistral-7B	3.76 ± 0.32	1.22 ± 0.01	−68%

INT4 throughput falls by 98% for both models.

4.5 Platform comparison

These measurements contrast sharply with reported GPU behaviour (Table 7). The same quantization approach produces opposite outcomes on different hardware, which demonstrates that the technique is platform dependent.

Table 7 CPU node versus GPU quantization effects

Platform	INT8 memory	INT8 speed	Hardware support
GPU (literature)	−50%	+50 to +100%	Hardware accelerator
CPU node (measured)	+48%	−68%	Software only

5 Discussion

5.1 Bottleneck and memory behaviour

The measurements form a single account of the node: it is limited by memory bandwidth, and that same limit explains why quantization behaves as it does.

Memory-bandwidth bottleneck. Utilization reaches 93–99% while throughput remains between 1.22 and 9.79 tokens per second at FP16 and INT8 for the 2.7B and 7B models. High utilization with low throughput indicates a bandwidth limit rather than a compute limit: the processor is occupied, but it is occupied waiting on weight

traffic from main memory rather than performing useful arithmetic. Single-stream decode reads the entire weight set once per generated token, which makes bandwidth the governing resource at this scale.

Quantization as diagnostic. INT8 reduces throughput by 57–68% despite halving weight storage, and INT4 collapses it by 98%. On a bandwidth-limited node, halving the bytes read per token should have made generation faster. That it does the opposite confirms that the tested CPU path lacks native low-precision execution: the bytes saved on the read are paid back, with interest, in conversion work before every operation.

Runtime memory exceeds weight size. The same absence of native low-precision execution governs the memory result. At FP16 the measured total tracks nominal weight size closely, so the familiar estimate of parameter count times bytes per parameter is adequate. Under quantization it is not: the measured total rises to roughly three times nominal weight size at INT8 and six times at INT4, because the bytes saved on weights are more than replaced elsewhere. For planning, a 7B model requires roughly 20 GB per node rather than the 14 GB implied by weight size.

Dual representation penalty. The 48% increase under INT8 is the cost of holding compressed weights and FP16 conversion buffers simultaneously, together with activations and an attention cache that remain in FP16 regardless of how the weights are stored. GPUs avoid this by computing directly on quantized values in dedicated hardware. The sizing model quantifies this penalty as a coefficient: overhead per billion parameters is effectively zero at FP16 but 1.89 GB at both INT8 and INT4. That the two quantized precisions share the same coefficient shows the penalty attaches to the quantized execution path rather than to the degree of compression.

Scaling implications. The 40–48% overhead is consistent across 1.1B, 2.7B, and 7B parameters, indicating a systematic property of the tested path rather than a per-model artifact. Since it holds across a 6× range, similar behaviour can be expected for larger models.

5.2 Implications for edge-tier deployment

The per-node figures translate directly into deployment decisions.

Node memory specification. A node intended to host a 7B model must be specified with roughly 20 GB available to the inference process, not the 14 GB implied by weight size. On a 32 GB node this leaves limited headroom for the operating system and colocated services, and it rules out hosting two such models on one node. Smaller models are correspondingly cheaper: Phi-2 requires about 5.4 GB. The sizing model of Section 4.3 generalizes this beyond the three models measured here: a candidate model and precision can be costed before any node is provisioned, and the multiplier form (1, 3 or 6 times weight size) is simple enough to apply during procurement.

Fleet sizing. Because a node saturates at one concurrent request, aggregate capacity is the node count multiplied by single-stream throughput. A tier of identical nodes serving Phi-2 at FP16 supplies approximately 9.79 tokens per second per node. Meeting a target aggregate rate is therefore a question of node count, and the per-node figures above give the multiplier.

The offload decision. Scheduling work across a tiered deployment is an optimization problem in its own right, addressed by schedulers that balance objectives such as completion time and energy across heterogeneous servers [1, 2]. Those schedulers require a cost model for each node, and the measurements here supply one for CPU-only language model serving. Sustained throughput determines which requests a node should retain, because generation cost accumulates linearly in output length. At the measured single-stream rates a 7B model needs about 27 s for a hundred-token reply, which exceeds interactive budgets, while the same node serving a 2.7B model needs about 10 s. Operators can therefore route by expected output length and model size rather than by request count alone.

Optimization must be validated on the node. The quantization result is the clearest warning. An operator applying INT8 to fit a model into a constrained node would obtain 48% more memory use and 68% less throughput, the opposite of the intended effect. Techniques adopted from GPU practice must be measured on the target node before being deployed across a fleet.

5.3 Platform dependency and root cause

These measurements show that GPU-derived optimization cannot be assumed to transfer. The approach that reduces GPU memory by 50% increases memory on this node by 48%.

The divergence has a hardware origin. Modern GPUs include units that compute directly on INT8 and INT4 data. The CPU uses general-purpose arithmetic units optimized for FP32 and FP64, and the tested libraries do not target the low-precision units that are present, so conversion to FP16 is performed in software before every operation. This explains both the memory increase and the throughput loss.

5.4 Limitations and future work

Platform scope. A single commodity edge CPU is evaluated. The quantization libraries tested do not use dedicated low-precision instructions, so quantized inference falls back to emulated paths. Server-class processors with hardware low-precision units (AMX, VNNI) can invert the overhead reported here; characterizing that class is a distinct problem left to future work.

Short context. The 11-token prompt represents common conversational use but not long-context workloads such as document analysis, where the KV-cache grows substantially.

No phase-level timing. The harness records total generation time but not individual token timestamps, so time to first token and inter-token latency are not measured and the prefill and decode phases are not separated here. Reported latencies are therefore aggregate. Resolving the phase split requires per-token instrumentation and is left to future work; it matters for interactive deployment, where perceived responsiveness depends on first-token latency rather than on average throughput.

Memory components not instrumented. Only process-level peak memory is measured. The decomposition in Table 3 is an estimate consistent with that total, so conclusions rest on total memory rather than on any individual component.

Early termination. Some replicates stopped before the 100-token target, most often for the 1.1B model. Throughput figures are therefore quoted for the 2.7B and 7B models, where all replicates ran to length; the 1.1B model contributes memory measurements, which are unaffected because peak memory is reached during model loading and the first forward passes.

Single quantization library. Testing used bitsandbytes, a GPU-oriented library. CPU-optimized formats such as GGML or ONNX Runtime may behave differently.

Single-node scope. Measurements characterize one node in isolation. Aggregate behaviour of a full edge tier, including request routing, inter-node load balancing, and contention for shared network or storage resources, is not measured here and follows from this per-node baseline.

Future work should extend these measurements to further CPU architectures including server-class processors with low-precision instruction support, test long-context behaviour to establish KV-cache scaling, compare CPU-optimized quantization libraries, and evaluate multi-node edge tiers built from characterized nodes.

6 Conclusion

This paper characterizes language model inference on a CPU-only edge node through measurement of three model sizes at three precision levels, establishing the per-node figures that distributed edge deployments require.

Nodes are memory-bandwidth limited: 93–99% utilization produces only 1.22 to 9.79 tokens per second, so a node is saturated by a single request. Runtime memory substantially exceeds nominal weight size, the remainder being activations, KV-cache, dequantization buffers, and framework overhead.

The measurements also show an unexpected result: quantization reduces GPU memory by 50–75% but increases memory on a CPU node by 40–48%, because the tested path lacks native low-precision execution and must hold both compressed weights and FP16 conversion buffers.

These measurements also yield a per-node sizing model. Runtime memory is predicted from parameter count and precision with 2.51% mean absolute error across all nine configurations, and reduces for practical use to multiplying nominal weight size by approximately 1 at FP16, 3 at INT8 and 6 at INT4. The calibrated overhead coefficient is effectively zero at FP16 and 1.89 GB per billion parameters at both quantized precisions, which locates the penalty in the quantized execution path rather than in the degree of compression.

For deployment, these figures are directly actionable. A node hosting a 7B model must be specified with about 20 GB rather than 14 GB. Since a node saturates at one request, aggregate capacity scales with node count at the measured single-stream rate. A 2.7B model sustains 9.79 tokens per second per node and supports interactive use, whereas GPU-oriented quantization libraries reduce node throughput by 57 to 98% and should not be applied without measurement on the target hardware.

Edge deployment of language models addresses real needs in privacy-sensitive, connectivity-constrained, and cost-limited settings. This characterization provides the per-node baseline on which multi-node edge inference tiers can be planned.

References

[1] Tiwari S, Beena BM (2024) A neighborhood inspired multiverse scheduler for energy and makespan optimized task scheduling for green cloud computing systems. IEEE Access 12:157272–157298. https://doi.org/10.1109/ACCESS.2024.3484388

[2] Beena BM, Ranga PC, Manideep TSS, et al (2025) A green cloud-based framework for energy-efficient task scheduling using carbon intensity data for heterogeneous cloud servers. IEEE Access 13:73916–73938. https://doi.org/10.1109/ACCESS.2025.3562882

[3] Gaur YN, Beena BM, Khanna M (2025) Performance optimization for matrix multiplication when offloading computation to GPU devices. In: Proceedings of Fourth International Conference on Computing and Communication Networks. Lecture Notes in Networks and Systems, Springer Nature Singapore, Singapore, p 89–100, https://doi.org/10.1007/978-981-96-3247-3_8

[4] Lalapura VS, Amudha J, Satheesh HS (2022) Recurrent neural networks for edge intelligence: a survey. ACM Computing Surveys 54(4):1–38. https://doi.org/10.1145/3448974

[5] Lalapura VS, Bhimavarapu VR, Amudha J, et al (2024) A systematic evaluation of recurrent neural network models for edge intelligence and human activity recognition applications. Algorithms 17(3):104. https://doi.org/10.3390/a17030104

[6] Jacob B, Kligys S, Chen B, et al (2018) Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 2704–2713

[7] Dettmers T, Lewis M, Belkada Y, et al (2022) LLM.int8(): 8-bit matrix multiplication for transformers at scale. In: Advances in Neural Information Processing Systems (NeurIPS)

[8] Frantar E, Alistarh D (2023) GPTQ: Accurate post-training quantization for generative pre-trained transformers. In: Proceedings of the International Conference on Learning Representations (ICLR)

[9] Gerganov G (2023) ggml: Tensor library for machine learning. Software library

[10] Microsoft (2023) ONNX runtime. Software library

[11] TensorFlow Team (2023) TensorFlow Lite. Software library

[12] Yu GI, Jeong JS, Kim GW, et al (2022) Orca: A distributed serving system for transformer-based generative models. In: Proceedings of the USENIX Symposium

on Operating Systems Design and Implementation (OSDI), pp 521–538

[13] Sheng Y, Zheng L, Yuan B, et al (2023) FlexGen: High-throughput generative inference of large language models with a single GPU. In: Proceedings of the International Conference on Machine Learning (ICML), pp 31094–31116

Statements and Declarations

Funding. The authors gratefully acknowledge Sri Mata Amritanandamayi Devi (Amma), Chancellor, Amrita Vishwa Vidyapeetham, for her inspiration and for providing financial support for the Article Processing Charges (APC) of this publication.

Conflict of interest. The authors declare that they have no conflict of interest.

Ethics approval and consent to participate. Not applicable.

Consent for publication. Not applicable.

Clinical trial number. Not applicable.

Data availability. The measurement data supporting the findings of this study are available from the corresponding author on reasonable request.

Materials availability. Not applicable.

Code availability. The profiling harness and analysis scripts are available from the corresponding author on reasonable request.

Author contribution. Yogesh Narayan Gaur designed and performed the experiments, analysed the data, and wrote the manuscript. Manju Khanna supervised the work and reviewed the manuscript. Both authors read and approved the final manuscript.