International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 7, Issue 3, 131-139, 2026
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V7I3P115

Original Article

Quantization and Compression Techniques for Model Deployment: Optimizing Neural Networks for Production

Rajalakshmi Srinivasaraghavan
Independent Researcher, USA.

Received On: 12/07/2026 ***Revised On: 19/08/2026*** ***Accepted On: 28/08/2026*** ***Published On: 03/09/2026***

Abstract - *Compression research reports compression ratios, and deployment engineering pays for something else. This paper reports a CPU benchmark study in which the binding constraint was neither model size nor latency but key-value cache memory, which grows linearly with sequence length and comes to dominate the inference memory budget well before weight storage does. Post-training INT8 quantization was applied to reduce that footprint, with latency treated as a secondary and unguaranteed benefit. The central finding concerns the gap between the compression ratio and the realized gain. Memory reduction followed the quantization ratio reliably. Runtime did not, because the software stack did not sustain low-precision execution end to end: several operators lacked native INT8 kernels, so tensors were repeatedly upcast to floating point and downcast back to INT8 between operations, and the conversion overhead consumed much of the arithmetic saving. The practical consequence is that a compression decision on CPU is governed by operator coverage across the execution path rather than by the nominal precision of the weights, and a technique that halves arithmetic width while doubling conversions can be a net loss. INT4 was considered and not pursued, on grounds of accuracy risk and the same coverage limitation in a more severe form. The paper also proposes a promotion process for moving a compressed model toward production against an uncompressed reference baseline. This is a benchmark feasibility study. No model was deployed to production or pilot, no quantitative result is reported, and Section IX states precisely what was not measured.*

Keywords - *Post-Training Quantization, INT8, KV-cache, Model Compression, CPU Inference, Mixed Precision, Operator Coverage, Inference Memory, Deployment Readiness, Neural Network Optimization.*

1. Introduction

Most compression work is reported against the metric it optimizes: a parameter count, a compression ratio, a theoretical reduction in multiply-accumulate operations. Deployment budgets are not denominated in any of those. A team adopting compression is paying for a specific constraint to be relieved, and which constraint that is determines whether a technique helps at all.

In the study reported here the binding constraint was key-value cache memory. In autoregressive transformer inference, the keys and values computed for every previously generated token are retained so they need not be recomputed. That cache grows linearly with sequence length, and beyond modest context lengths it becomes the dominant term in the inference memory budget, exceeding the storage occupied by the weights themselves. This is what limited how efficiently larger models could be evaluated and scaled in the benchmark environment.

That framing determines everything that follows. Quantization was applied to reduce cache storage. Latency and compute improvements were treated as secondary gains, welcome if they arrived and not the object of the exercise. Techniques that reduce weight storage without touching the cache were correspondingly less interesting, regardless of the compression ratios they report.

The central empirical finding concerns the distance between a compression ratio and a realized benefit, and the two halves of that finding diverged sharply. Memory behaved as the arithmetic predicted. Reducing the precision of cached tensors reduced the memory they occupied, reliably and in proportion. untime did not. The expected speed-up was substantially eroded because the software stack did not sustain low-precision execution across the whole graph. Several operators had no native INT8 implementation, so tensors were upcast to floating point to traverse them and downcast back to INT8 afterwards. Those conversions are pure overhead. They perform no useful arithmetic, they occur at every coverage gap along the execution path, and their cost is invisible in any analysis that reasons from precision ratios alone.

The practical claim of this paper follows from that asymmetry: on CPU, a compression decision is governed by operator coverage across the execution path, not by the nominal precision of the weights. A technique that halves arithmetic width while introducing conversions at a dozen boundaries can be a net loss, and the quantization ratio gives no warning of it.

Section II positions the work and states the boundary against the author's prior CPU performance work. Section III develops the KV-cache constraint analytically. Sections IV

to VII cover the method, the mixed-precision boundary problem, the quality assessment, and why INT4 was declined. Section VIII proposes a promotion process. Sections IX to XI state what was not measured, the limitations, and the conclusion.

2. Background and positioning

2.1. What Each Technique Actually Reduces

The compression literature is mature and its main families are well characterized. Their relevance here is determined by which part of the memory budget each one touches.

- Quantization: reduces the numerical precision of tensors. Foundational treatments established integer-arithmetic-only inference for efficient deployment [1] and set out the practical design space of per-tensor and per-channel scaling, calibration, and layer sensitivity [2], [3]. Surveys consolidate the field [4]. Applied to weights it reduces model storage; applied to the key-value cache it reduces the term that dominates at long sequence length.
- Pruning: removes parameters or structures, reducing weight storage and potentially compute [5], [6]. It does not reduce the KV-cache, because the cache is a function of sequence length, layer count, and head geometry rather than of weight density.
- Knowledge distillation: trains a smaller student to reproduce a larger teacher's behaviour [7]. It reduces the cache indirectly by reducing model dimensions, but it requires a training pipeline rather than a post-hoc transformation.
- Low-rank factorization: decomposes weight matrices, again acting on weights rather than on the cache.
- For a KV-cache-bound workload, this taxonomy is decisive: only quantization of the cached tensors, or a change of model geometry, acts on the binding constraint. Pruning and factorization optimize a term that was not the bottleneck.

2.2. Post-Training Quantization for Transformer Inference

Within quantization, post-training methods are attractive for deployment study because they require no retraining. Work on transformer-scale models has established that outlier activations complicate naive integer quantization, and has proposed mixed-precision decomposition [8], error-compensating weight quantization [9], and activation smoothing to make ranges more tractable [10].

The KV-cache specifically has become its own line of work, with methods that quantize cached keys and values to low bit widths while managing the accuracy consequences [11], [12]. That literature confirms the constraint framing adopted here: the cache is treated as the scaling bottleneck for long-context inference, and the analysis of inference cost structure at scale supports the same conclusion [13].

2.3. Boundary against the Author's Prior Work

The author has previously published on CPU-side performance of AI workloads, including inference endpoint scheduling for large language model serving on heterogeneous infrastructure [14].

The boundary is as follows. That work concerns where and when inference work is placed, treating the model as given. The present paper concerns what the model is made of numerically, and specifically what happens to the memory and runtime budget when the precision of cached tensors is reduced. Neither restates the other. Where this paper touches scheduling, it does so only to note that the promotion process in Section VIII assumes a serving path exists, which is the subject of that prior work rather than of this one.

Further prior work by the author on CPU optimization and performance characterization is related in subject area. It is not cited here because the venues concerned fall outside the citation standard applied to this paper, and the present work does not depend on their content.

2.4. The Gap

The quantization literature reports accuracy against bit width, generally on accelerators with mature low-precision kernel support. What is less often reported is the case where the arithmetic saving is real and the execution stack cannot sustain it, so the benefit is consumed by format conversion at operator boundaries. That is the case this paper documents, and on CPU it is the common case rather than the exception.

3. The binding constraint: KV-cache memory

3.1. Why the Cache Dominates

The size of the key-value cache is determined by model geometry and sequence length rather than by parameter count. For a transformer decoder it is given by

cache_bytes = 2 x layers x kv_heads x head_dim x sequence_length x batch_size x bytes_per_element

with the factor of two accounting for keys and values. Every term except sequence length is fixed once the model is chosen. Sequence length is not fixed, and it is the term that grows during generation.

The consequence is a crossover. At short sequence lengths, weight storage dominates and the intuition that a smaller model is a cheaper model holds. As sequence length grows, the cache term grows linearly while weight storage stays constant, and beyond a certain context the cache is the larger consumer. Past that point, compressing weights optimizes the smaller of the two terms.

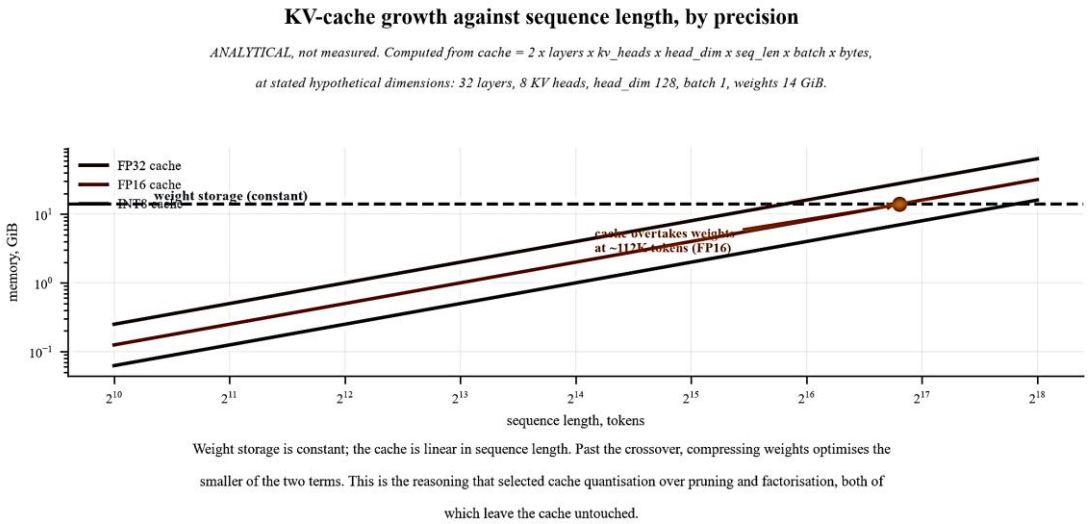

Fig 1: Key-Value Cache Size against Sequence Length at Three Precisions, with Weight Storage Shown for Comparison. Values are computed from the Closed-Form Expression above Using Stated Hypothetical Model Dimensions. These are Analytical Values, Not Measurements. The Crossover Point, Where the Cache Overtakes Weight Storage, is the Property that determines which Compression Technique is worth Applying

Table 1: Cache Size by Element Precision, as a Ratio of the FP32 Baseline. This is Arithmetic on the Expression above, Not a Measured Result. The Ratios are Exact; whether they are Realizable End to End is the Subject of Section V

Precision	Bytes per element	Relative cache size	What it acts on
FP32	4	1.00	Baseline
FP16 or BF16	2	0.50	Cache and weights
INT8	1	0.25	Cache and weights
INT4	0.5	0.125	Cache and weights, with materially higher accuracy risk

3.2. Why This Reframes the Technique Choice

Once the cache is identified as the constraint, the technique comparison changes. Table I summarizes what was considered against what was genuinely applied.

Table 2: Techniques Considered Against Techniques Applied. The Third Column is the Selection Criterion for this Study: For a KV-Cache-Bound Workload, a Technique that Does Not Reduce Cache Memory Does Not Address the Binding Constraint, Whatever Compression Ratio It Reports on Weights

Technique	Acts on	Reduces KV-cache	Status in this study
Post-training INT8 quantization	Weights and cached tensors	Yes, in proportion to precision	**Genuinely applied.** The subject of this paper
INT4 quantization	Weights and cached tensors	Yes, further	Considered, not pursued. See Section VII
Quantization-aware training	Weights and activations	Yes, via precision	Not implemented. Requires a training pipeline
Structured or unstructured pruning	Weights	No	Not implemented. Does not act on the constraint
Knowledge distillation	Model geometry	Indirectly, via smaller dimensions	Not implemented. Requires training
Low-rank factorization	Weight matrices	No	Not implemented. Does not act on the constraint

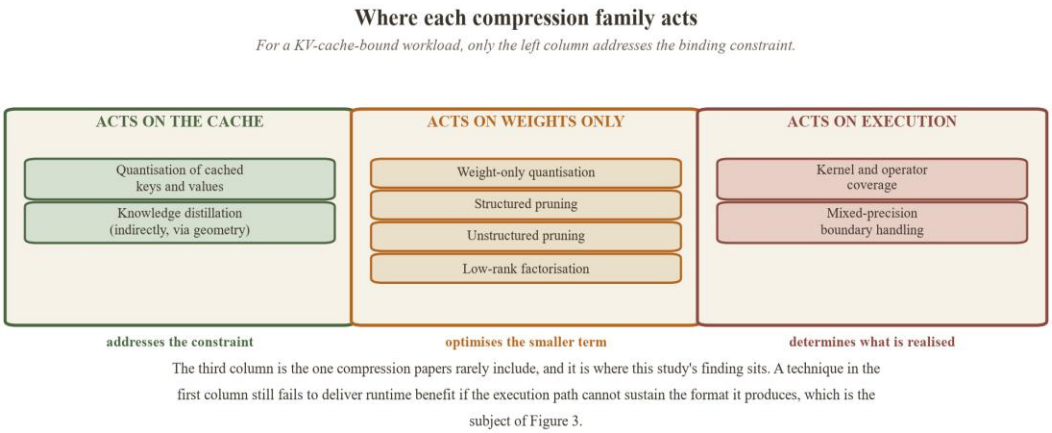

Fig 2: Where Each Compression Family Acts Within the Inference Memory Budget. The Distinction that Matters for a Long-Context Workload is the Middle Column: Pruning and Factorization Reduce Weight Storage and Leave the Cache untouched, so they optimize the Term that Was Not Binding

4. Method

Post-training INT8 quantization was applied to a floating-point baseline model and evaluated on CPU-based benchmark systems. The comparison in every case was the quantized variant against its own floating-point baseline on the same hardware, since the object was the effect of the transformation rather than an absolute performance figure.

INT8 was selected over more aggressive alternatives on two grounds. It produced a clear reduction in memory footprint, and it remained comparatively stable under CPU inference, where the execution-path limitations of Section V apply with less severity than at lower bit widths.

What this method does not include, stated explicitly. The design decisions that a complete quantization study would report, namely the choice between per-tensor and per-channel scaling, the composition and size of the calibration set, and the identification of layers that had to be retained at higher precision, were not systematically recorded in this study. They are acknowledged here as a gap rather than reconstructed after the fact. Section IX treats this as one of the study's principal limitations, because those choices materially affect the accuracy outcome and their absence bounds what the accuracy discussion in Section VI can claim.

5. The Mixed-Precision Boundary Problem

This section reports the study's central finding.

5.1. Memory Delivered, Runtime Did Not

On the CPU benchmark environment, the theoretical gains from quantization translated into reduced memory footprint reliably and into runtime speed-up only partially. The dependable benefit was the reduction in KV-cache memory. Latency improvement was limited, and the limitation was not a property of the arithmetic.

5.2. The Mechanism

The cause was incomplete low-precision execution support in the software stack. Although parts of the model could be represented in INT8, several operators had no native INT8 kernel available. Execution therefore could not remain in low precision across the whole graph.

At each such operator, the tensor was upcast to floating point, the operator executed in floating point, and the result downcast back to INT8 for the next stage. Those conversions perform no useful computation. They consume memory bandwidth and instruction issue slots purely to change representation, and they recur at every coverage gap along the path.

5.3. Why the Quantization Ratio Gives No Warning

The important property of this failure is that nothing in the compression metrics predicts it. The quantization ratio is a property of the numeric format. The realized speed-up is a property of how many operators along the execution path can consume that format natively, which is a property of the compiler and kernel library rather than of the model.

Two deployments of an identically quantized model on stacks with different operator coverage can therefore produce materially different runtime outcomes, and neither is anomalous.

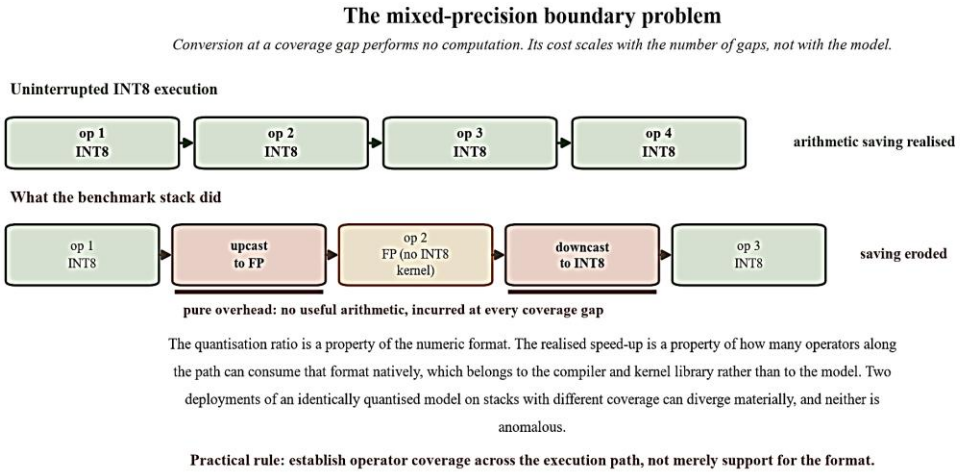

Fig 3: The Mixed-Precision Boundary Problem. The Upper Path is Uninterrupted Int8 Execution, In Which the Arithmetic Saving Is Realized. The Lower Path is What the Benchmark Stack Actually Did: An Operator without an Int8 Kernel Forces an Upcast and a Matching Downcast, And Each Pair is Overhead Attributable to Coverage rather Than to the Model

Table 3: Predicted Against Realized Outcomes. The First Row Is The Reason The Study Concluded That Quantization Was Worthwhile For Its Stated Purpose. The Remaining Three Are the Reason the Runtime Claim Was Not Made

Predicted From Quantization Ratio	Realized On The CPU Benchmark Stack	Cause Of Divergence
Memory footprint reduced in proportion to precision	Delivered, reliably, and largest for KV-cache	Storage is a property of representation, which the format changes directly
Arithmetic cost reduced in proportion to precision	Partially delivered	Only realized on operators with native INT8 kernels
End-to-end latency reduced accordingly	Substantially eroded	Upcast and downcast at every coverage gap; pure overhead
Benefit portable across hardware and stacks	Not portable	Realized gain depends on operator coverage, not on the model

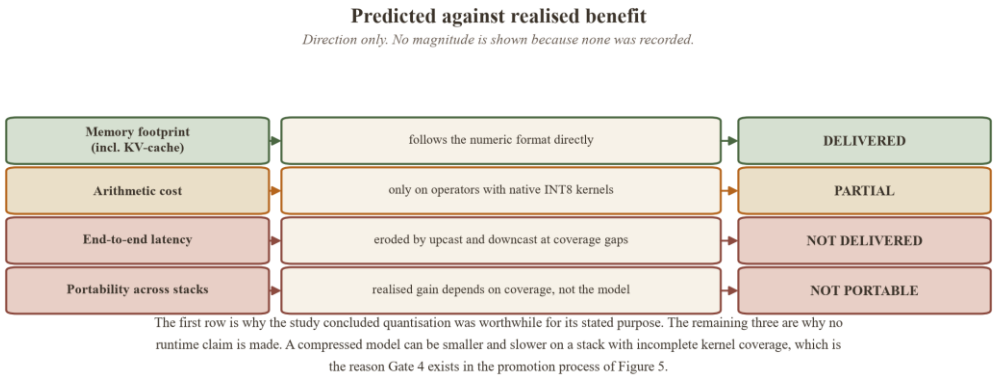

Fig 4: The Divergence between Predicted and Realized Benefit across the Two Dimensions. Memory Follows the Format; Runtime Follows Kernel Coverage. The Figure Is A Qualitative Summary Of The Finding Described Above And Carries No Measured Quantity, Since None Was Recorded.

5.4. The Practical Rule

The rule this supports for practitioners is narrow and testable. Before adopting a precision reduction on CPU, establish operator coverage across the intended execution path, not merely support for the format. A model that is representable in INT8 and executable in INT8 at only some operators will pay conversion cost at every boundary between the two, and that cost scales with the number of gaps rather than with the model.

6. Quality Assessment and Its Limits

Model quality was assessed by comparing the output behaviour of the quantized variant against the floating-point baseline on a representative benchmark set. Inputs were

selected to reflect the same inference scenarios used to study cache behaviour, including longer-sequence cases, since those are where cache pressure and any precision-related instability would both be expected to appear.

The analysis considered whether quantization introduced noticeable changes in responses, and whether behaviour became unstable under typical or extended inputs. The limits of that approach should be stated plainly, because they bound the paper's accuracy claims. No formal accuracy metric was computed against a labeled evaluation set. No analysis was performed of degradation concentrated in rare classes, edge inputs, or particular input segments, which is the class of regression that aggregate comparison is least able to detect. A comparison of output behaviour can establish that a quantized model has not become obviously unstable. It cannot establish that accuracy has been preserved, and this paper does not claim that it has.

7. Why INT4 Was Not Pursued

INT4-style compression was considered and deliberately not taken further. Two reasons, in order of weight.

- Accuracy risk: The reduction from INT8 to INT4 is a substantially larger perturbation, and the assessment approach available in this study, described in Section VI, was not sensitive enough to characterize its consequences responsibly. Adopting a more aggressive precision while lacking the means to measure its cost would have produced a claim the study could not support.
- The coverage problem, intensified: The execution-path limitation of Section V applies more severely at lower bit widths. Native INT4 kernel coverage on the CPU stack was thinner than INT8 coverage, so the expected additional saving would have been offset by more frequent conversion, and the implementation complexity of managing that path was greater.

The general finding is worth separating from the specific decision. More aggressive low-precision settings were not rejected because the compression was insufficient. They were rejected because the expected efficiency gains were not matched by equally reliable execution behaviour on the available stack, which is the same asymmetry documented in Section V appearing at a more demanding operating point.

8. A Promotion Process from Benchmark toward Production

No model in this study was promoted. This section proposes the process by which one should be, and it is offered as a design contribution rather than as a report of practice. The organizing principle is that the uncompressed model remains the reference version throughout. The compressed variant is a candidate measured against it, not a replacement assumed to be equivalent.

Table 4: Proposed Promotion Gates. Gate 4 Exists Because Of Section V: A Quantized Model Can Be Smaller And Slower On A Stack With Incomplete Kernel Coverage, And A Process That Checks Size Without Checking Realized Latency Will Promote It.

Gate	What is checked	Against what	Release blocked if
1. Offline equivalence	Output drift between compressed variant and baseline on representative workloads	Uncompressed baseline, same inputs	Drift exceeds the predefined acceptance threshold
2. Stability	Behaviour under typical and extended-sequence inputs	Baseline behaviour	Instability appears that the baseline does not exhibit
3. Resource	Memory footprint, including cache behaviour at target sequence lengths	Declared budget	The constraint that motivated compression is not actually relieved
4. Latency	Median and tail response time on the target stack	Baseline, same hardware	Realized latency regresses, per the Section V mechanism
5. Versioning	Compressed artifact is versioned and traceable to the baseline it derives from	Model registry	Provenance to a specific baseline version cannot be established

Acceptance thresholds are defined before evaluation rather than after, since a threshold chosen once results are visible is not a gate. Release is blocked unless the candidate falls within them.

After release, monitoring tracks output deviation from the baseline, failure rates, latency, and resource usage. Rollback is to the retained uncompressed baseline, which is why keeping it as the reference version, rather than retiring it once a compressed variant ships, is a precondition of the whole process.

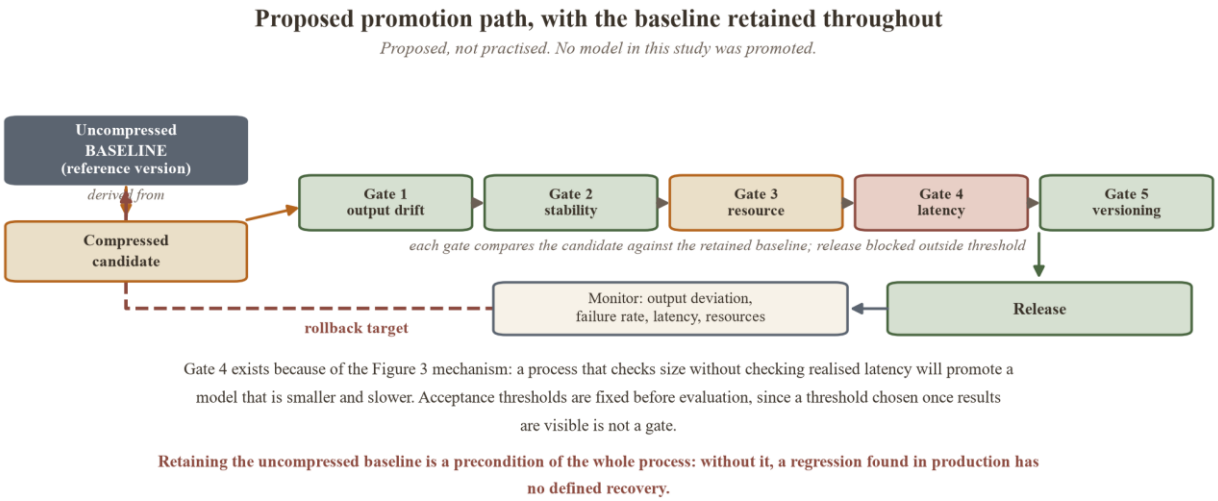

Fig 5: The Proposed Promotion Path. The Uncompressed Baseline Persists As Both The Comparison Reference Before Release And The Rollback Target After It. The Dashed Return Path Is What Makes the Process Reversible; Without a Retained Baseline, a Quality Regression Discovered In Production Has No Defined Recovery

9. What Was Not Measured

This section is stated at length because the paper's credibility depends on the boundary being explicit rather than inferred.

No quantitative result is reported: Specifically absent are model size before and after compression, median and tail latency, throughput, memory footprint in absolute units, cost per inference, and accuracy delta for any technique. The benchmarking supported a directional comparison and was not instrumented to the completeness those figures would require.

What the benchmarking did support: Is the directional conclusion reported in Section V: post-training INT8 quantization produced a clear reduction in memory footprint, particularly for cache-related inference memory, while runtime gains were more limited and contingent on compiler and kernel support.

More aggressive approaches were explored qualitatively only and were not validated with a metric set sufficient to support quantitative claims. Section VII reports the decision, not a measured comparison.

Method detail was not systematically recorded, as stated in Section IV: scaling granularity, calibration set composition, and layer-level precision retention were not captured. This is the gap most likely to affect reproducibility. Environment detail cannot be disclosed. The benchmarking used restricted hardware, so system specifications, runtime configuration, and kernel versions are unavailable for publication. Findings are therefore reported as mechanisms rather than as configurations, which is a real limitation on independent verification and is acknowledged as such.

Table 5: Evidence Status by Quantity. The Middle Column Is Deliberately Conservative: Directionally Observed Means The Direction Was Consistent Across The Benchmark Runs Performed, Not That A Magnitude Was Established.

Quantity	Status	Consequence for the paper
Memory footprint reduction	Directionally observed, not quantified	Supports a qualitative claim only
Runtime speed-up	Directionally observed as limited, not quantified	Supports the mechanism in Section V, not a magnitude
Accuracy delta	Not computed against a labeled set	No accuracy preservation claim is made
Model size before and after	Not recorded	Cannot be reported
Throughput, cost per inference	Not recorded	Cannot be reported
Scaling granularity, calibration, layer retention	Not recorded	Bounds reproducibility
Hardware and runtime configuration	Restricted	Bounds independent verification

10. Limitations

- Benchmark only: Nothing here was deployed to production or pilot. The promotion process in Section VIII is proposed and untested.
- One technique: Post-training INT8 quantization is the only method genuinely applied. Statements about pruning, distillation, quantization-aware training, and factorization are positioning, not findings.
- One hardware class: The findings concern CPU-based systems. The mixed-precision boundary problem is a property of operator coverage, so its severity on other hardware, particularly accelerators with mature low-precision kernels, would be expected to differ and was not examined.
- Accuracy assessment is comparative, not metric-based: As Section VI states, output-behaviour comparison cannot substitute for measured accuracy on a labeled set, and no rare-class or segment analysis was performed.
- Method detail is incomplete: Scaling granularity, calibration procedure, and layer-level precision decisions were not recorded, which limits reproducibility independently of the disclosure restriction.
- No reviewer was named: No technical reviewer was identified to check the draft for accuracy or for material that must not be published, which the disclosure restrictions make advisable before submission.

11. Conclusion

The argument of this paper is that a compression decision should be made against the constraint that is actually binding, and then verified against what the execution stack can deliver rather than what the arithmetic promises.

In this study the binding constraint was key-value cache memory, which grows linearly with sequence length and overtakes weight storage well before long-context inference becomes interesting. That framing selects quantization of cached tensors and deselects pruning and factorization, not because those techniques are weak but because they reduce a term that was not the bottleneck.

The finding that generalizes beyond this study is the divergence between the two halves of the expected benefit. Memory reduction followed the format directly and reliably. Runtime reduction did not, because several operators lacked native INT8 kernels and execution could not remain in low precision across the graph, so tensors were repeatedly upcast and downcast at coverage gaps. That conversion cost performs no computation, scales with the number of gaps rather than with the model, and is invisible to any analysis reasoning from quantization ratios. The practical rule is to establish operator coverage across the intended execution path before adopting a precision reduction, since a model representable in INT8 but executable in INT8 only in places will pay at every boundary between the two.

INT4 was declined on the same reasoning applied at a more demanding operating point, compounded by an accuracy risk this study lacked the instrumentation to characterize responsibly.

What this paper does not establish should be as clear as what it does. No quantitative result is reported, no accuracy claim is made, and no model reached production or pilot. Section IX lists the absent quantities individually rather than gesturing at limitations, and the promotion process in Section VIII is a proposal awaiting the deployment that would test it. The contribution is a constraint-first framing, a documented failure mode of CPU quantization that compression ratios do not predict, and a process for promoting a compressed model without assuming it is equivalent to the baseline it replaces.

References

[1] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 2704-2713. doi: 10.1109/CVPR.2018.00286

[2] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," arXiv:1806.08342, 2018. [Online]. Available: https://arxiv.org/abs/1806.08342

[3] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A White Paper on Neural Network Quantization," arXiv:2106.08295, 2021. [Online]. Available: https://arxiv.org/abs/2106.08295

[4] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," arXiv:2103.13630, 2021. [Online]. Available: https://arxiv.org/abs/2103.13630

[5] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," arXiv:1510.00149, 2015. [Online]. Available: https://arxiv.org/abs/1510.00149

[6] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," arXiv:1803.03635, 2018. [Online]. Available: https://arxiv.org/abs/1803.03635

[7] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," arXiv:1503.02531, 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[8] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale," arXiv:2208.07339, 2022. [Online]. Available: https://arxiv.org/abs/2208.07339

[9] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate Post-Training Quantization for

Generative Pre-trained Transformers," arXiv:2210.17323, 2022. [Online]. Available: https://arxiv.org/abs/2210.17323

[10] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models," arXiv:2211.10438, 2022. [Online]. Available: https://arxiv.org/abs/2211.10438

[11] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, "KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization," arXiv:2401.18079, 2024. [Online]. Available: https://arxiv.org/abs/2401.18079

[12] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, "KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache," arXiv:2402.02750, 2024. [Online]. Available: https://arxiv.org/abs/2402.02750

[13] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently Scaling Transformer Inference," arXiv:2211.05102, 2022. [Online]. Available: https://arxiv.org/abs/2211.05102

[14] R. Srinivasaraghavan, "Intelligent Inference Endpoint Scheduling for Heterogeneous Large Language Model Serving," International Journal of Computer Information Systems and Industrial Management Applications, 2026. doi: 10.70917/ijcisim-2026-4862.