Quantization Benchmarking with LLaMA

Final Project Report for Advanced Computer Architecture I

Damir Sarsenov
Advanced Computer Architecture I, Fall 2025
California State University Long Beach
Email: `damir.sarsenov01@student.csulb.edu`
Submission date: December 2025

***Abstract*—This project studies two complementary approaches to low-precision inference for Large Language Models (LLMs): *GPTQ*, a post-training, weight-only quantization method that compresses Transformer weights to 3–4 bits using approximate second-order information, and *Tender*, an algorithm–hardware co-design that uses tensor decomposition and runtime requantization to create efficient low-precision execution paths. GPTQ shows that large GPT-style models can be compressed to 3–4-bit weights with negligible accuracy loss and end-to-end speedups over FP16 on modern GPUs [1]. Tender demonstrates that carefully structured decomposed quantization with power-of-two-spaced scales can further improve throughput and accuracy with only minimal modifications to tensor hardware [2].**

We analyze these techniques through the lens of LLaMA-class models (8B and 70B parameters) as representative open LLMs. Our focus is on four architecture-relevant metrics: model quality, time-to-first-token (TTFT), tokens/s, and memory/energy per token. The main contribution of this work is a lightweight, architecture-aware benchmark protocol and prototype harness that future work can adopt to report system-level metrics alongside model quality, making quantization results for LLaMA-class models more directly comparable.

I. Introduction

Large Language Models (LLMs) such as the LLaMA family have become central workloads in modern datacenters. Their parameter counts (e.g., 8B and 70B parameters for recent LLaMA variants) and long sequence lengths stress:

- DRAM/HBM capacity and bandwidth (weights plus KV cache),
- compute throughput (billions of MACs per token),
- latency (time-to-first-token (TTFT) and per-token delay),
- energy per token (dominated by memory traffic and underutilized compute).

Quantization is one of the few architectural knobs that directly targets all four of these concerns: by reducing the precision of weights and/or activations, it shrinks the memory footprint and can increase arithmetic intensity per byte moved.

In this project we focus on two representative approaches:

- **GPTQ (ICLR 2023)** – a post-training, weight-only quantization method that uses approximate second-order (Hessian-based) information to minimize reconstruction error for each weight block, enabling accurate 3–4-bit quantization for very large Transformer models [1].
- **Tender (ISCA 2024)** – an algorithm–hardware co-design that introduces tensor decomposition and power-of-two-spaced scales, allowing partial sums to be combined via bit shifts instead of repeated dequantize/quantize operations [2].

Contribution. Rather than proposing a new quantization method, this paper:

1) synthesizes the architectural ideas behind GPTQ and Tender,
2) maps their reported results to a common set of LLaMA-class metrics (quality, TTFT, tokens/s, memory, energy),
3) proposes a concrete benchmark protocol and a small prototype harness that can be used in future work to evaluate quantization schemes for LLaMA-class models.

II. Background and Related Work

A. LLaMA-class Models and Architectural Challenges

LLaMA-class models are autoregressive Transformers with billions of parameters and sequence lengths in the hundreds or thousands. Their large parameter counts push GPU memory capacity, leading to:

- model or tensor parallelism across multiple devices to fit weights,
- heavy HBM traffic every decoding step,
- substantial energy consumption dominated by data movement.

These characteristics make LLaMA-class models an ideal case study for quantization and other architectural optimizations.

B. Quantization for Transformer Inference

Quantization reduces numerical precision to lower memory footprint and improve effective compute throughput. Common design axes include:

- weight-only vs. weight+activation quantization,
- per-tensor vs. per-channel vs. per-group scaling,
- post-training quantization (PTQ) vs. quantization-aware training (QAT),
- uniform vs. non-uniform quantization, including learnable or power-of-two scales.

For large LLMs, post-training, weight-only methods are attractive because they avoid expensive retraining and can be applied to off-the-shelf checkpoints.

C. GPTQ: Hessian-aware Post-training Quantization

GPTQ is a one-shot, post-training, weight-only quantization algorithm [1]. It assumes a small calibration dataset and a full-precision model. For each weight block, GPTQ:

- constructs a quadratic approximation of the loss using an approximate Hessian,
- selects quantized weights that minimize the resulting reconstruction error, subject to a low-bit constraint (e.g., 3–4 bits),
- uses computational tricks to avoid inverting large Hessian matrices explicitly.

GPTQ demonstrates that GPT-style models up to 175B parameters can be compressed to 3–4-bit weights with negligible loss in language modeling quality, while 2-bit regimes show more visible degradation [1].

D. Tender: Tensor Decomposition and Requantization Avoidance

Tender targets the overhead of requantization in fully integer pipelines [2]. It:

- decomposes each weight matrix into multiple components,
- assigns power-of-two-spaced scales to these components,
- accumulates partial sums at runtime by aligning them via bit shifts instead of dequantize/quantize operations.

Because the scale factors are powers of two, scale changes can be implemented as shifts, which are cheap in hardware. The method leads to higher accuracy and throughput than prior low-precision schemes at similar effective bitwidths, with only small hardware extensions [2].

III. Design and Methodology

A. Problem Statement and Research Questions

The overarching goal is to understand how GPTQ and Tender, as two different quantization philosophies, impact LLaMA-class inference under architecture-relevant metrics.

We formulate the following research questions:

RQ1: How close can 3–4-bit weight-only post-training quantization stay to full-precision quality on LLaMA-class models?

RQ2: For realistic accelerator hardware, how much end-to-end speedup (TTFT, tokens/s) can be attributed to quantization, as reported by GPTQ and Tender?

RQ3: What additional gains or complexities arise when modest hardware modifications (as in Tender) are allowed, compared to a purely software-only method (GPTQ)?

RQ4: What minimal benchmark protocol is necessary to compare quantization schemes fairly on LLaMA-class workloads?

B. Benchmarking Framework Design

We design a framework centered on four key metrics:

- **Quality:** perplexity or task accuracy vs. a full-precision baseline.
- **Performance:** TTFT and tokens/s.
- **Memory:** weights-only footprint plus peak memory usage.
- **Energy:** energy per token, where measurements are available.

For each precision regime (e.g., FP16/BF16, W4 with GPTQ, a Tender-style configuration), we conceptually run the same workload and record these metrics, controlling for software stack and hardware platform.

C. Benchmark Protocol Pseudocode

Algorithm 1 summarizes the proposed benchmark protocol that a future implementation could realize.

Algorithm 1 Protocol for Quantization Benchmarking on LLaMA-class Models

```
Require: Model family $\mathcal{M}$ (e.g., LLaMA 8B, 70B), precision
    schemes $\mathcal{P}$ (e.g., FP16, GPTQ W4, Tender-style), task set $\mathcal{T}$
 1: for each model $m \in \mathcal{M}$ do
 2:     for each precision scheme $p \in \mathcal{P}$ do
 3:         Quantize or configure $m$ under $p$
 4:         Measure weights-only memory footprint
 5:         for each task $t \in \mathcal{T}$ do
 6:             Fix prompts, seeds, and batch size
 7:             Record TTFT and tokens/s (cold and warm)
 8:             Compute quality metric (perplexity or task score)
 9:             Optionally, record board power and compute energy per token
10:         end for
11:     end for
12: end for
13: Compare precision schemes along all metrics for each model
```

This protocol is intentionally simple, but it is sufficient to capture the trade-offs exposed by GPTQ and Tender.

IV. Implementation Details

Our prototype implementation of this protocol is a small Python harness (`quant_bench_llama.py`) that:

- loads a causal LLaMA-style model and tokenizer from Hugging Face,
- runs the model under several backends (`fp16`, `bnb4`, `gptq4`),
- measures TTFT proxy, average latency, throughput (tokens/s), and perplexity on WikiText-2 prompts,
- writes results to structured JSON files under a `results/` directory.

On a CPU-only machine we use `TinyLlama/TinyLlama-1.1B-Chat-v1.0` as a small LLaMA-style model. On a GPU with sufficient memory, the same harness can be run on `meta-llama/Meta-Llama-3-8B` or `meta-llama/Meta-Llama-3-70B` by changing the `--model-id` argument.

TABLE I
PROTOTYPE BENCHMARK RESULTS FOR TINYLLAMA-1.1B-CHAT (FP16, CPU-ONLY).

Model	TTFT [ms]	Latency [ms]	Thru. [tok/s]	PPL
TinyLlama-1.1B	2673.57	16552.97	1.93	10.56

In a full implementation of this project, we would deploy the benchmarking protocol using:

- **Models:** LLaMA-class checkpoints (e.g., 8B and 70B parameters).
- **Frameworks:** A standard deep learning stack (e.g., PyTorch) and a serving framework capable of measuring TTFT and tokens/s reliably.
- **Quantization libraries:** A GPTQ implementation for weight-only 3–4-bit quantization, and a research implementation of Tender or a simplified variant for decomposition-based quantization.
- **Hardware:** At least one modern GPU with tensor/integer units (e.g., A100-class) to reflect the hardware used in GPTQ and Tender.

Practical challenges include:

- integrating custom integer kernels for GPTQ and Tender into the serving stack,
- ensuring reproducible measurements by fixing seeds and controlling concurrent load,
- dealing with GPU memory constraints when running 70B models at full precision vs. low-bit regimes.

Beyond the small-scale prototype experiment reported in Section V, we do not run full 8B/70B LLaMA models ourselves; instead we rely on the published GPTQ and Tender results for large-scale numbers.

V. EXPERIMENTAL SETUP AND RESULTS

In this section we first report a small prototype experiment on TinyLlama-1.1B using our harness, and then organize the *reported* results from GPTQ and Tender and connect them to the proposed benchmarking dimensions.

A. Prototype Evaluation on TinyLlama-1.1B (CPU)

To demonstrate the benchmark harness end-to-end, we ran a small-scale experiment on a CPU-only machine using `TinyLlama/TinyLlama-1.1B-Chat-v1.0` as a LLaMA-style model. We used the FP16 backend, four WikiText-2 validation prompts (`max_samples = 4`), and up to 32 new tokens per prompt (`max_new_tokens = 32`). The resulting metrics, taken directly from the JSON file `results_fp16_16bit.json` in the artifact, are summarized in Table I.

As expected, the CPU-only setting yields low throughput and high latency, but the harness successfully produces TTFT, latency, throughput, and perplexity metrics from a single script and writes them into structured JSON files. On a GPU with sufficient memory, replacing the model ID with `meta-llama/Meta-Llama-3-8B` or `meta-llama/Meta-Llama-3-70B` would exercise the same code path at full LLaMA scale.

TABLE II
BACK-OF-THE-ENVELOPE WEIGHTS-ONLY MEMORY FOOTPRINT FOR LLAMA-CLASS MODELS.

Model	FP16 (16-bit)	INT8 (8-bit)	INT4 (4-bit)
LLaMA 8B	$\sim$16 GB	$\sim$8 GB	$\sim$4 GB
LLaMA 70B	$\sim$140 GB	$\sim$70 GB	$\sim$35 GB

B. Back-of-the-Envelope Memory Model

For weights-only memory, a simple model is:

$$\text{bytes} \approx \text{params} \times \frac{\text{bits}}{8}. \qquad (1)$$

Using 8B and 70B LLaMA-class models as proxies and assuming a single precision for all weights, we obtain Table II. KV cache, activations, optimizer states, and metadata are excluded.

C. Reported GPTQ Results

GPTQ reports that:

- 3–4-bit weight-only quantization can closely match full-precision perplexity on standard language modeling benchmarks for GPT/OPT-style models up to 175B parameters [1].
- With specialized INT3/INT4 kernels, end-to-end speedups of more than $3\times$ over FP16 are achievable on A100- and A6000-class GPUs under favorable batching and sequence-length settings [1].

From the perspective of our protocol, GPTQ occupies the “W4, weights-only” column in Table II and typically yields:

- negligible quality loss versus FP16,
- multi-$\times$ improvements in tokens/s,
- significant reduction in weights-only memory and bandwidth.

D. Reported Tender Results

Tender reports that:

- its decomposition plus power-of-two scale policy yields higher or comparable accuracy relative to previous low-precision schemes at similar bitwidths [2],
- by eliminating frequent requantization, it achieves up to about $2.6\times$ end-to-end speedups over FP16 baselines [2].

In our framework, Tender represents a co-designed point in the space: it targets low-precision execution with decomposed weights and small hardware extensions that remove a key overhead (requantization).

VI. ANALYSIS AND DISCUSSION

A. Comparative Summary

Table III summarizes the high-level trade-offs between GPTQ and Tender.

The TinyLlama-1.1B CPU prototype results in Table I are far from the LLaMA-3 8B/70B regimes we ultimately care

TABLE III
QUALITATIVE COMPARISON OF GPTQ AND TENDER FOR LLAMA-CLASS INFERENCE.

Method	Precision Regime	Key Idea	Strengths	Caveats
GPTQ [1]	W3–W4 (3–4-bit weights-only)	Hessian-aware one-shot PTQ	Strong quality-vs-size trade-off; drop-in for existing accelerators; public code and kernels; no hardware changes required.	Speedups depend on kernel maturity and framework integration; activations and accumulations remain high precision, limiting maximum theoretical savings.
Tender [2]	Low-precision with decomposed tensors	Power-of-two scale spacing; runtime requantization avoidance	Higher accuracy/throughput vs. prior low-precision schemes; minimal but principled hardware extensions; good utilization of integer pipelines.	Requires hardware and software co-design; exact reproduction may need specialized code and modified hardware assumptions.

about, but they confirm that the harness behaves as intended: it produces TTFT, latency, throughput, and perplexity metrics in a reproducible way that can be scaled up on GPU hardware.

From an architectural perspective:

- GPTQ is an attractive *baseline* for any LLaMA quantization study because it is software-only and compatible with existing accelerators.
- Tender is a compelling *direction* when modest hardware changes are possible; it demonstrates how small extensions can unlock additional benefits beyond weight-only PTQ.

B. Energy Considerations

Neither GPTQ nor Tender is primarily an energy-measurement paper, but both reduce data movement, which typically lowers energy per token. Qualitatively:

- weight-only 4-bit quantization directly reduces HBM traffic for weights, which often dominates energy consumption,
- better pipeline utilization (less time stalled on memory or conversions) improves performance per watt,
- overheads such as repeated dequantize/quantize operations can erode some of these benefits; Tender explicitly targets this overhead and thus has a favorable energy story if implemented in hardware.

C. Design Implications for Future Work

The main design implication of this analysis is that quantization research should be evaluated at *system level*, not only in terms of model quality:

- reporting TTFT, tokens/s, memory footprint, and energy per token alongside quality,
- controlling for serving framework and kernel versions,
- using comparable workloads across precision regimes.

The benchmark protocol in Algorithm 1 provides a concrete template to do so.

VII. CONCLUSION AND FUTURE WORK

Weight-only post-training quantization (GPTQ) and algorithm–hardware co-design (Tender) represent two complementary strategies for efficient LLaMA inference:

- GPTQ shows that 3–4-bit weight-only quantization can drastically reduce memory footprint and enable multi-$\times$ speedups while maintaining quality close to full precision, given mature integer kernels and solid implementations [1].
- Tender shows that modest hardware changes, combined with structured decomposed quantization, can avoid requantization overhead, improve utilization of integer pipelines, and achieve additional speedups and accuracy improvements over earlier low-precision schemes [2].

From an Advanced Computer Architecture perspective, these works underscore a broader theme: the bottleneck in LLM inference is often not pure compute, but the interplay between precision, memory bandwidth, and hardware execution paths.

Future work that would deepen this project includes:

- implementing GPTQ and a subset of Tender ideas on LLaMA-class models using the proposed benchmark protocol,
- extending the comparison to other quantization methods (e.g., activation-aware or group-wise schemes),
- measuring energy per token directly using power instrumentation to move from qualitative to quantitative energy claims,
- exploring how these quantization strategies interact with serving-level optimizations such as speculative decoding, continuous batching, and KV cache sharding.

Standardizing TTFT, tokens/s, memory, and energy reporting—along the lines of this protocol—would make it much easier for architects to compare future quantization proposals for LLaMA-class models and beyond.

REFERENCES

[1] E. Frantar, S. Ben Ashkboos, T. Hoefler, and D. Alistarh, “GPTQ: Accurate post-training quantization for generative pre-trained transformers,” in *Proc. International Conference on Learning Representations (ICLR)*, 2023.

[2] J. Lee, H. Lee, and J. Sim, “Tender: Accelerating large language models via tensor decomposition and runtime requantization,” in *Proc. International Symposium on Computer Architecture (ISCA)*, 2024.