

Comprehensive Evaluation of Quantization Methods for Edge LLM Deployment

Zahraa Selim, Menna Hamed, Wesam Ahmed, Sohyla Said, Sara Basheer, and Rami Zewail

Computer Science and Engineering Department

Egypt-Japan University of Science and Technology (E-JUST)

Alexandria, Egypt

{zahraa.selim, menna.hamed, rami.zewail}@ejust.edu.eg

I. METHODOLOGY

A. Base Model and Hardware

This study utilizes **Mistral-7B-v0.1** [1] as the baseline model, selected for its parameter efficiency and strong performance profile. All experiments are conducted on a single **NVIDIA Tesla T4 GPU (16GB VRAM)** to simulate realistic edge deployment scenarios. The software environment includes PyTorch 2.x, Transformers 4.x, and quantization-specific libraries (bitsandbytes, AutoGPTQ, AutoAWQ, HQQ).

B. Compression Techniques

We evaluate five distinct 4-bit quantization strategies, each representing different algorithmic approaches:

1) NF4 (NormalFloat 4-bit)

NF4 utilizes the quantile-based NormalFloat data type, which is information-theoretically optimal for zero-centered normal distributions. Double quantization is employed to further minimize memory footprint by quantizing the quantization constants themselves [?].

Configuration:

- Quantization type: NF4 (NormalFloat4)
- Double quantization: Enabled
- Compute dtype: float16
- Group size: Per-tensor

Implementation:

```
1 from transformers import BitsAndBytesConfig
2 import torch
3
4 nf4_config = BitsAndBytesConfig(
5     load_in_4bit=True,
6     bnb_4bit_quant_type="nf4",
7     bnb_4bit_use_double_quant=True,
8     bnb_4bit_compute_dtype=torch.float16
9 )
10
11 model = AutoModelForCausalLM.from_pretrained(
12     "mistralai/Mistral-7B-Instruct-v0.1",
13     quantization_config=nf4_config,
```

```
14     device_map="auto"
15 )
```

2) GPTQ

GPTQ employs layer-wise quantization minimizing reconstruction error via inverse Hessian information [?]:

$$\min_{\hat{W}} \|WX - \hat{W}X\|_2^2 \quad (1)$$

where W is the weight matrix and X represents calibration inputs.

Configuration:

- Bits: 4
- Group size: 128
- Dataset: C4 (calibration)
- Kernel: ExLlamaV2

Implementation:

```
1 from auto_gptq import AutoGPTQForCausalLM
2
3 model = AutoGPTQForCausalLM.from_quantized(
4     "TheBloke/Mistral-7B-Instruct-v0.1-GPTQ",
5     device="cuda:0",
6     use_safetensors=True
7 )
```

3) AWQ (Activation-aware Weight Quantization)

AWQ protects salient weights based on activation magnitudes, preserving the top 1% most impactful weights with per-channel scaling [?].

Configuration:

- Bits: 4
- Group size: 128
- Zero point: True
- Version: GEMM

Implementation:

```
1 from awq import AutoAWQForCausalLM
2
```

This work was conducted at Egypt-Japan University of Science and Technology (E-JUST).

```

3 model = AutoAWQForCausalLM.from_quantized(
4     "TheBloke/Mistral-7B-Instruct-v0.1-AWQ",
5     fuse_layers=True,
6     safetensors=True
7 )

```

```

16 ar.quantize_and_save(
17     output_dir=OUTPUT_DIR,
18     format="auto_round"
19 )

```

4) HQQ (Half-Quadratic Quantization)

HQQ performs fast, calibration-free quantization optimizing a custom loss function with on-the-fly parameter optimization [?].

Configuration:

- Bits: 4
- Group size: 64
- Axis: 1 (row-wise)
- Compute dtype: float16

Implementation:

```

1 from hqq.models.hf.base import AutoHQQHModel
2 from hqq.core.quantize import BaseQuantizeConfig
3
4 quant_config = BaseQuantizeConfig(
5     nbits=4, group_size=64, axis=1
6 )
7
8 model = AutoHQQHModel.from_pretrained(
9     "mistralai/Mistral-7B-Instruct-v0.1",
10    torch_dtype=torch.float16
11 )
12 model.quantize_model(
13     quant_config=quant_config, device='cuda'
14 )

```

5) AutoRound

AutoRound employs adaptive quantization with iterative optimization for per-layer parameter tuning.

Configuration:

- Bits: 4
- Group size: 128
- Samples: 64
- Iterations: 50
- Batch size: 4
- Learning rate: 5e-3

Implementation:

```

1 from auto_round import AutoRound
2
3 ar = AutoRound(
4     model=MODEL_PATH,
5     scheme="W4A16",
6     bits=4,
7     group_size=128,
8     nsamples=64,
9     iters=50,
10    lr=5e-3,
11    seqlen=1024,
12    batch_size=4,
13    amp_dtype=torch.float16
14 )
15

```

C. Evaluation Metrics

We assess compressed models across two primary dimensions: computational efficiency and task performance.

1) Efficiency Metrics

a) Latency Measurements

- **Average Latency:** Mean time per generated token (ms)
- **Time-to-First-Token (TTFT):** Initial response latency
- **Prefill vs. Decode:** Separate measurement of prompt processing vs. autoregressive generation

b) Throughput and Memory

- **Throughput:** Tokens per second (tok/s)
- **Peak Memory:** Maximum GPU memory (MB)
- **Model Size:** Disk storage requirements (GB)

c) Computational Efficiency

- **Model FLOPs Utilization (MFU):** Percentage of theoretical peak hardware FLOPs achieved:

$$MFU = \frac{\text{Achieved FLOPs/s}}{\text{Peak Hardware FLOPs/s}} \times 100\% \quad (2)$$

- **Energy Consumption:** Estimated energy per token (mJ):

$$E = (P_{TDP} - P_{idle}) \times t_{inference} \quad (3)$$

where $P_{idle} \approx 0.3 \times P_{TDP}$

2) Performance Benchmarks

We evaluate models using established language modeling benchmarks:

a) Language Modeling

Perplexity: Measured on WikiText-2 test set using sliding window evaluation (stride 512):

$$PPL(W) = \exp \left(-\frac{1}{N} \sum_{i=1}^N \ln P(w_i | w_{<i}) \right) \quad (4)$$

b) Core Task Benchmarks

- **HellaSwag** (0-shot): Commonsense reasoning via sentence completion
- **ARC-Easy** (0-shot): Grade-school science questions
- **ARC-Challenge** (0-shot): Challenge-level scientific reasoning
- **GSM8K** (8-shot): Grade-school math word problems
- **MMLU** (5-shot): Multi-domain knowledge across 57 subjects

- **HumanEval** (0-shot): Python code generation (pass@1)

All evaluations use the Language Model Evaluation Harness [?] with consistent hyperparameters for reproducibility.

II. EXPERIMENTAL RESULTS

A. Computational Efficiency Analysis

Table I presents comprehensive efficiency metrics on Tesla T4 hardware.

TABLE I
EFFICIENCY METRICS ON NVIDIA TESLA T4

Method	Disk (GB)	Peak Mem (MB)	TTFT (ms)	Decode (ms/T)	TPut (T/s)	Energy (mJ/T)
FP16	13.49	7013	68.84	61.36	15.84	3086
NF4	3.74	1859	177.16	79.09	12.30	3975
AWQ	3.87	1869	79.25	77.22	12.79	3855
HQQ	3.74	4721	664.73	649.84	1.51	32421
GPTQ	3.87	4415	1113.00	1117.76	0.88	55703

Key Findings:

- 1) **Hardware Compatibility Crisis:** GPTQ and HQQ exhibit severe performance degradation on Tesla T4, with throughputs below 2 tok/s and energy consumption exceeding 30,000 mJ/token. This represents a **10-18x slowdown** compared to NF4/AWQ, rendering them impractical for this hardware.
- 2) **NF4 vs. AWQ Trade-off:** NF4 achieves 12.30 tok/s with 3975 mJ/token, while AWQ achieves 12.79 tok/s with 3855 mJ/token. AWQ provides marginal efficiency gains (3% better energy), but as shown in Section IV-B, this comes at the cost of task performance.
- 3) **Memory Efficiency:** All 4-bit methods achieve approximately 3.6x compression (13.49 GB → 3.7-3.9 GB), validating theoretical compression ratios. However, peak memory usage varies significantly, with HQQ and GPTQ requiring 2.5x more runtime memory due to kernel overhead.

B. General Task Performance

Table II presents performance across reasoning benchmarks.

Performance Summary:

Task-Specific Analysis:

- 1) **Knowledge Retrieval Robustness:** ARC-Challenge and ARC-Easy maintain near-FP16 performance (0.58 vs 0.58 for NF4), indicating that factual knowledge retrieval is preserved under quantization.
- 2) **Mathematical Reasoning Degradation:** GSM8K experiences 25% accuracy drop (0.36 → 0.27), suggesting multi-step reasoning is more sensitive to quantization than pattern-matching tasks.

- 3) **Code Generation Stability:** HumanEval maintains perfect parity (0.05 across all methods), likely due to the task's low baseline difficulty for 7B models.
- 4) **Perplexity Mismatch:** Despite similar perplexity scores, methods show different task performance profiles, confirming that perplexity alone is insufficient for evaluating practical model quality.

III. DISCUSSION

A. Hardware-Algorithm Compatibility as Primary Factor

Our most critical finding is that **hardware compatibility dominates compression effectiveness**. While theoretical compression ratios are identical (3.6x for all 4-bit methods), practical performance varies by 18x on Tesla T4.

Hypothesis: GPTQ and HQQ rely on INT4 tensor cores and optimized GEMM kernels introduced in Ampere architecture. On Turing GPUs (Tesla T4), these operations fall back to inefficient FP16 emulation, causing severe overhead.

Implication: For edge deployment on consumer/previous-generation GPUs, **distribution-based quantization (NF4) is the only viable option**. Kernel-dependent methods should be reserved for latest-generation hardware.

B. Task-Dependent Sensitivity

Quantization impacts tasks differentially:

- **Preserved:** Knowledge retrieval (ARC), commonsense reasoning (HellaSwag)
- **Degraded:** Mathematical reasoning (GSM8K -25%)
- **Stable:** Code generation (HumanEval)

This suggests that quantization affects **deep reasoning chains** more than **pattern matching**. For applications requiring complex multi-step logic, either fine-tuning or hybrid precision strategies may be necessary.

C. The NF4 vs. AWQ Trade-off

AWQ offers marginal efficiency gains (3% lower energy) but shows higher perplexity degradation (+5.32% vs +1.80% for NF4). For general-purpose deployment, **NF4 provides better performance preservation** despite slightly higher energy consumption.

However, AWQ may be preferable for:

- Latency-critical applications where 3% speedup matters
- Tasks where perplexity degradation is acceptable
- Scenarios prioritizing energy efficiency over quality

D. Deployment Guidelines

Based on our comprehensive analysis, we provide prescriptive recommendations:

TABLE II
PERFORMANCE COMPARISON ACROSS BENCHMARKS

Method	Perplexity (↓)	HellaSwag (0-shot)	ARC-Easy (0-shot)	ARC-Challenge (0-shot)	GSM8K (8-shot)	MMLU (5-shot)	HumanEval (0-shot)
FP16	12.79	0.72	0.76	0.58	0.36	1.00	0.05
NF4	13.02	0.70	0.75	0.58	0.27	0.55	0.05
GPTQ	12.85	0.68	0.75	0.60	—	—	0.05
AWQ	13.47	—	—	—	—	—	—
HQQ	13.50	0.69	0.72	0.50	—	—	—

TABLE III
AVERAGE ACCURACY AND DEGRADATION

Method	Avg Acc	PPL Inc	Acc Drop	Tasks
FP16	0.57	—	—	6
NF4	0.52	+1.80%	-0.05	6
GPTQ	0.52	+0.47%	-0.05	6
AWQ	—	+5.32%	—	6
HQQ	—	—	—	6

TABLE IV
QUANTIZATION METHOD SELECTION GUIDE

Scenario	Recommended Method
Turing GPUs (T4, RTX 20-series)	NF4 (only viable option)
Ampere+ GPUs (A100, RTX 30/40)	AWQ or GPTQ (test both)
Math-heavy applications	NF4 + targeted fine-tuning
Knowledge retrieval	Any 4-bit method
Maximum energy efficiency	AWQ (if compatible)
General-purpose deployment	NF4 (best balance)

IV. CONCLUSION

This study provides the first comprehensive hardware-aware benchmarking of modern quantization methods on consumer-grade edge hardware. Our key contributions include:

- 1) **Hardware Compatibility Analysis:** Demonstrating that algorithm-hardware compatibility is the primary determinant of practical performance, with kernel-dependent methods experiencing 10-18x slowdown on Turing GPUs.
- 2) **Task-Specific Evaluation:** Establishing that quantization impacts tasks differentially, with mathematical reasoning showing 25% degradation while knowledge retrieval remains robust.
- 3) **Prescriptive Guidelines:** Identifying NF4 as the optimal method for Tesla T4 and similar hardware, achieving 3.6x compression with minimal performance loss and superior hardware compatibility.

For Turing-generation hardware, NF4 represents the **only viable quantization method**, achieving stable 12.30 tok/s

throughput with reasonable energy consumption (3975 mJ/token). While AWQ offers marginal efficiency gains, NF4's superior task performance and hardware stability make it the recommended choice for general-purpose edge deployment.

Future Work: Investigating hybrid precision strategies where critical layers (attention heads for reasoning tasks) retain higher precision while feedforward networks are aggressively quantized may resolve the mathematical reasoning degradation observed in our study.

ACKNOWLEDGMENTS

This research was conducted at Egypt-Japan University of Science and Technology (E-JUST). We thank the Anthropic team for access to computational resources.

Bibliography section - uncomment and fill in when ready

REFERENCES

- [1] A. Q. Jiang et al., “Mistral 7B,” *arXiv preprint arXiv:2310.06825*, 2023.