Compression Techniques for Mistral-7B: A Comprehensive Evaluation of Efficiency and Performance

Zahraa Selim Menna Hamed Wesam Ahmed Sohyla Said Sara Basheer

Under the supervision of
Prof. Rami Zewail

Department of Computer Science and Engineering,
Egypt-Japan University of Science and Technology (E-JUST)

1 Experimental Setup

1.1 Overview

This study conducts a comprehensive comparative evaluation of multiple compression techniques applied to the Mistral-7B model. Our experimental protocol follows a two-phase approach: (1) compress the base model and evaluate compression impact, and (2) fine-tune the compressed models on downstream tasks and re-evaluate to measure performance recovery. All experiments are conducted under resource-constrained environments to reflect realistic deployment scenarios.

1.2 Environment and Resources

All experiments were conducted on cloud-based platforms with the following specifications:

- **Platforms:** Kaggle and Google Colab free-tier instances
- **Hardware:** NVIDIA Tesla T4 GPU (16GB VRAM)
- **Software:** PyTorch 2.x, Transformers 4.x, bitsandbytes, AutoGPTQ, AutoAWQ
- **Base Model:** Mistral-7B (FP16 baseline, 13.49 GB model size)

These resource constraints (limited VRAM and compute) reflect the target deployment scenario for compressed models and ensure our findings are applicable to practical edge and consumer-grade hardware settings.

1.3 Compression Techniques

1.3.1 Quantization Methods

We evaluate four state-of-the-art post-training quantization techniques:

- **NF4 (4-bit NormalFloat):** Uses an information-theoretically optimal data type for normally distributed weights, implemented via bitsandbytes with double quantization enabled.

  Parameters:

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

  Implementation:

```python
from transformers import BitsAndBytesConfig, AutoModelForCausalLM
import torch

# Configure NF4 quantization
nf4_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.float16
)

# Load model with quantization
model = AutoModelForCausalLM.from_pretrained(
    "mistralai/Mistral-7B-Instruct-v0.1",
    quantization_config=nf4_config,
    device_map="auto"
)
```

Listing 1: NF4 Quantization with BitsAndBytes

- **GPTQ:** Layer-wise quantization that minimizes reconstruction error using Hessian information, applied at 4-bit precision with group size of 128. We use TheBloke's pre-quantized model to avoid the computationally expensive quantization process.

  Parameters:

  - Bits: 4
  - Group size: 128
  - Dataset: C4 (used during quantization)
  - Activation order: Optimized via Hessian

  Implementation:

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

# Load pre-quantized GPTQ model
model = AutoModelForCausalLM.from_pretrained(
    "TheBloke/Mistral-7B-Instruct-v0.1-GPTQ",
    device_map="auto",
    trust_remote_code=False,
    revision="main"
)

tokenizer = AutoTokenizer.from_pretrained(
    "TheBloke/Mistral-7B-Instruct-v0.1-GPTQ",
    use_fast=True
)
```

Listing 2: GPTQ Quantization (Pre-quantized Model)

- **AWQ (Activation-aware Weight Quantization):** Protects salient weights based on activation magnitudes, using 4-bit quantization with per-channel scaling. Similar to GPTQ, we use a pre-quantized model.

  Parameters:

  - Bits: 4
  - Group size: 128
  - Zero point: True
  - Version: GEMM (optimized matrix multiplication)

  Implementation:

```python
from awq import AutoAWQForCausalLM
from transformers import AutoTokenizer

# Load pre-quantized AWQ model
model = AutoAWQForCausalLM.from_quantized(
    "TheBloke/Mistral-7B-Instruct-v0.1-AWQ",
    fuse_layers=True,
    trust_remote_code=False,
    safetensors=True
)

tokenizer = AutoTokenizer.from_pretrained(
    "TheBloke/Mistral-7B-Instruct-v0.1-AWQ",
    trust_remote_code=False
)
```

Listing 3: AWQ Quantization (Pre-quantized Model)

- **HQQ (Half-Quadratic Quantization):** Fast quantization method optimizing a custom loss function, configured for 4-bit weights with optimized zero-point placement. This method quantizes the model on-the-fly during loading.

  Parameters:

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

  Implementation:

```python
from hqq.engine.hf import HQQModelForCausalLM
from transformers import AutoTokenizer
import torch

# Configure HQQ quantization
quant_config = {
    'nbits': 4,
    'group_size': 64,
    'axis': 1,
    'compute_dtype': torch.float16
}

# Load and quantize with HQQ
model = HQQModelForCausalLM.from_pretrained(
    "mistralai/Mistral-7B-Instruct-v0.1",
    quantization_config=quant_config,
    device='cuda'
)

tokenizer = AutoTokenizer.from_pretrained(
    "mistralai/Mistral-7B-Instruct-v0.1"
)
```

Listing 4: HQQ Quantization (On-the-fly)

All quantization methods target 4-bit precision to achieve similar compression ratios (approximately 3.6x) for fair comparison. NF4 and HQQ quantize on-the-fly during model loading, while GPTQ and AWQ use pre-quantized checkpoints from TheBloke's repository for efficiency.

1.4 Evaluation Metrics and Benchmarks

Our evaluation framework assesses compressed models across three dimensions: computational efficiency, task performance, and retrieval-augmented generation capabilities.

1.4.1 Efficiency Metrics

We measure computational efficiency through the following metrics:
Latency Measurements:

- **Average Latency:** Mean time per generated token (ms) measured across multiple inference runs with warmup iterations to ensure stable GPU states
- **Time to First Token (TTFT):** Initial response latency measuring the time from prompt submission to first token generation, critical for interactive applications
- **Prefill vs. Decode Latency:** Separate measurement of prompt processing time (prefill) and autoregressive generation time (decode) to identify optimization bottlenecks

Table 1: Latency measurement parameters

Parameter	Default Value	Description
num_warmup	3	Warmup iterations before measurement
num_runs	10	Number of measurement iterations
max_new_tokens	128	Maximum tokens to generate per prompt
prompts	8 prompts	List of test prompts for benchmarking

Throughput and Memory:

- **Throughput:** Sustained generation rate measured in tokens per second, averaged over extended generation sequences
- **Peak Memory:** Maximum GPU memory allocated during inference (MB), captured using CUDA memory profiling
- **Model Size:** Disk storage requirements (GB) including all parameters and buffers
- **Memory Efficiency:** Ratio of model size to peak memory usage, indicating memory overhead beyond model parameters (e.g., activations, KV cache)

Table 2: Throughput and memory measurement parameters

Parameter	Default Value	Description
num_runs	10	Number of measurement iterations
max_new_tokens	128	Tokens to generate per run
batch_size	1	Batch size for evaluation
measure_batch_throughput	false	Test multiple batch sizes
batch_sizes	[1, 2, 4, 8]	Batch sizes to test (if enabled)

Computational Efficiency:

- **Model FLOPs Utilization (MFU):** Percentage of theoretical peak hardware FLOPs achieved during inference, calculated as $\text{MFU} = \frac{\text{Achieved FLOPs/s}}{\text{Peak Hardware FLOPs/s}} \times 100\%$, where achieved FLOPs is the product of FLOPs per token and throughput
- **Energy Consumption:** Estimated energy per token (mJ) based on device Thermal Design Power (TDP) and measured latency, using the formula $E = (P_{\text{TDP}} - P_{\text{idle}}) \times t$, where P_{idle} is assumed to be 30% of TDP

Table 3: Computational efficiency parameters

Parameter	Default Value	Description
device_tdp_watts	Auto-detected	Device thermal design power
idle_power_ratio	0.3	Fraction of TDP at idle (30%)
peak_tflops	Auto-detected	Hardware peak TFLOPs (FP16)

1.4.2 Performance Benchmarks

We evaluate model quality using established language modeling benchmarks, organized by capability:
Language Modeling:

- **Perplexity:** Measured on WikiText-2 test set using sliding window evaluation with stride 512 to assess next-token prediction quality. Lower perplexity indicates better language understanding.

Table 4: Perplexity evaluation parameters

Parameter	Default Value	Description
dataset	wikitext	HuggingFace dataset name
dataset_config	wikitext-2-raw-v1	Dataset configuration
split	test	Dataset split to evaluate
num_samples	100	Number of samples to process
max_length	512	Maximum sequence length
stride	512	Sliding window stride (null=no sliding)
batch_size	1	Batch size for processing

Selected Core Tasks:
All core tasks use the Language Model Evaluation Harness [**?**] with consistent hyperparameters for reproducibility. We report accuracy (or pass@1 for code tasks) normalized to [0,1].

Table 5: Core task benchmark parameters

Task	Few-Shot	Metric	Description
HellaSwag	0	acc_norm	Commonsense reasoning via sentence completion in everyday scenarios
ARC-Easy	0	acc_norm	Grade-school level science question answering
ARC-Challenge	0	acc_norm	Challenge-level scientific reasoning questions
GSM8K	8	exact_match	Grade-school math word problems with step-by-step reasoning
MMLU	5	acc	Multi-domain knowledge across 57 academic subjects
HumanEval	0	pass@1	Python code generation evaluated on test case pass rate

Table 6: LM-Eval harness global parameters

Parameter	Default Value	Description
batch_size	1	Global batch size for all tasks
limit	null	Limit samples per task (null=all)
random_seed	1234	Random seed for reproducibility

Original Mistral-7B Benchmarks:

Table 7: Mistral-7B complete benchmark suite

Category	Tasks	Few-Shot	Metric
Commonsense Reasoning	HellaSwag	0	acc_norm
	Winogrande	0	acc
	PIQA	0	acc_norm
	SIQA	0	acc
	OpenbookQA	0	acc_norm
	ARC-Easy	0	acc_norm
	ARC-Challenge	0	acc_norm
	CommonsenseQA	0	acc
World Knowledge	NaturalQuestions	5	exact_match
	TriviaQA	5	exact_match
Reading Comprehension	BoolQ	0	acc
	QuAC	0	f1
Math	GSM8K	8 (maj@8)	exact_match
	MATH	4 (maj@4)	exact_match
Code	HumanEval	0	pass@1
	MBPP	3	pass@1
Aggregate Benchmarks	MMLU	5 (57 tasks)	acc
	BBH	3 (23 tasks)	acc
	AGI Eval	3-5	acc

1.4.3 RAG Evaluation

For retrieval-augmented generation, we evaluate both retrieval quality and answer generation using a custom question-answering dataset.

Retrieval Quality Metrics:

Table 8: Retrieval quality metrics and parameters

Metric	Description
Context Sufficiency	Fraction of queries where retrieved contexts contain sufficient information (80% token overlap threshold)
Context Precision	Relevance of retrieved chunks measured by query-context token overlap
Context Coverage	Fraction of answer tokens present in retrieved contexts
Retrieval Consistency	Standard deviation of retrieval scores, indicating stability
Precision@K	Fraction of top-K retrieved items that are relevant
Recall@K	Fraction of relevant items in top-K retrieved
F1@K	Harmonic mean of Precision@K and Recall@K
MRR	Mean reciprocal rank of first relevant item
MAP	Mean average precision across all queries

Table 9: Retrieval evaluation parameters

Parameter	Default Value	Description
top_k	3	Number of chunks to retrieve
k_values	[1, 3, 5, 10]	K values for precision@k, recall@k
similarity_threshold	0.3	Minimum similarity score threshold
relevance_token_threshold	0.3	Token overlap threshold for relevance
sufficiency_token_threshold	0.8	Token overlap threshold for sufficiency

Answer Generation Metrics:

Table 10: Answer generation metrics and parameters

Metric	Description
Exact Match (EM)	Binary correctness: perfect normalized string match
F1 Score	Token-level precision-recall harmonic mean
Answer Relevance	Query-answer token overlap (measures if answer addresses question)
Faithfulness	Token containment: fraction of answer tokens in retrieved context
ROUGE-1/2/L	N-gram overlap (unigram, bigram) and longest common subsequence
BERTScore	Semantic similarity using contextual BERT embeddings (F1)
BLEU	Translation-style matching with smoothing

Table 11: Answer generation parameters

Parameter	Default Value	Description
max_new_tokens	128	Maximum tokens in generated answer
temperature	0.3	Sampling temperature (lower=deterministic)
top_p	0.9	Nucleus sampling threshold
repetition_penalty	1.15	Penalty for repeated tokens
normalize_whitespace	true	Normalize whitespace in comparisons
case_sensitive	false	Case-sensitive matching
remove_punctuation	false	Remove punctuation before comparison
rouge_use_stemmer	true	Use Porter stemmer for ROUGE
bertscore_lang	en	Language for BERTScore

RAG Efficiency:

Table 12: RAG efficiency metrics

Metric	Description
Retrieval Time	Average time to retrieve top-k contexts (ms)
RAG Generation Time	Time to generate answers with retrieved context (ms)
No-RAG Generation Time	Baseline generation time without context (ms)
RAG Throughput	Generation speed with context (tokens/sec)
No-RAG Throughput	Generation speed without context (tokens/sec)
Generation Speedup	Ratio of no-RAG to RAG generation time
F1 Improvement	Delta between RAG and no-RAG F1 scores
EM Improvement	Delta between RAG and no-RAG exact match scores

Table 13: RAG evaluation dataset parameters

Parameter	Default Value	Description
num_questions	10	Number of QA pairs to evaluate
dataset_path	null	Path to custom QA dataset (JSON)
compare_no_rag	true	Compare with no-RAG baseline
save_detailed_responses	false	Save individual Q&A responses

All RAG metrics are averaged over the evaluation dataset sampled from a technical documentation corpus, with retrieval configured to return top-3 chunks per query by default.

2 Results

2.1 Overview

We present a comprehensive comparison of compression techniques across three dimensions: efficiency gains, performance preservation, and RAG capabilities. Our analysis focuses on understanding the trade-offs between model size reduction, inference speed, and task performance for each compression method.

2.2 Compression Efficiency Analysis

2.2.1 Quantization Methods

Table 14: Efficiency comparison of quantization methods on Tesla T4 GPU

Method	Size (GB)	Memory (MB)	Latency (ms/tok)	TTFT (ms)	Throughput (tok/s)	MFU (%)	Energy (mJ/tok)	Prefill (ms)	Decode (ms/tok)
FP16	13.49	7010.65	62.48	—	15.97	2.38	3061.60	—	—
NF4	3.74	1859.11	84.14	—	11.60	1.73	4122.85	—	—
GPTQ	—	—	—	—	—	—	—	—	—
AWQ	—	—	—	—	—	—	—	—	—
HQQ	—	—	—	—	—	—	—	—	—

Table 15: Memory and compression details for quantization methods

Method	Total Params	Bits per Param	Memory Efficiency	KV Cache (MB)	Compression Ratio	Memory Reduction
FP16	7.24B	16.0	—	—	1.00x	1.00x
NF4	7.24B	4.0	—	—	3.61x	3.77x
GPTQ	7.24B	—	—	—	—	—
AWQ	7.24B	—	—	—	—	—
HQQ	7.24B	—	—	—	—	—

2.3 Performance Preservation Analysis

2.3.1 Quantization Methods

Table 16: Performance comparison of quantization methods 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	0.05
NF4	13.02	0.70	0.75	0.58	0.27	0.55	0.05
GPTQ	—	—	—	—	—	—	—
AWQ	—	—	—	—	—	—	—
HQQ	—	—	—	—	—	—	—

Table 17: Average accuracy and performance degradation for quantization methods

Method	**Average Accuracy**	**Perplexity Increase**	**Accuracy Drop**	**Tasks Evaluated**
FP16	—	—	—	4
NF4	—	+1.80%	—	4
GPTQ	—	—	—	0
AWQ	—	—	—	0
HQQ	—	—	—	0

2.4 RAG Performance Analysis

2.4.1 Quantization Methods

Table 18: RAG answer quality evaluation of quantization methods

Method	F1	EM	Faithfulness	Relevance	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore F1
FP16	0.217	0.0	0.559	0.109	0.279	0.091	0.197	0.658
NF4	0.133	0.0	0.333	0.065	—	—	0.137	0.437
GPTQ	—	—	—	—	—	—	—	—
AWQ	—	—	—	—	—	—	—	—
HQQ	—	—	—	—	—	—	—	—

Table 19: RAG vs no-RAG comparison for quantization methods

Method	No-RAG F1	No-RAG EM	F1 Improvement	EM Improvement	Avg Answer Length (RAG)	Avg Answer Length (No-RAG)
FP16	19.04	0.0	+0.011	—	—	—
NF4	—	—	-0.029	—	—	—
GPTQ	—	—	—	—	—	—
AWQ	—	—	—	—	—	—
HQQ	—	—	—	—	—	—

Table 20: Context retrieval quality across quantization methods

Method	Sufficiency	Precision	Coverage	Consistency (std)	Avg Score	Avg Chunks Retrieved	Avg Context Length
FP16	0.796	0.634	0.756	0.095	0.8	3	1308.5
NF4	0.756	0.634	0.716	0.095	—	—	—
GPTQ	—	—	—	—	—	—	—
AWQ	—	—	—	—	—	—	—
HQQ	—	—	—	—	—	—	—

Table 21: IR-style retrieval metrics for quantization methods

Method	P@1	P@3	P@5	R@1	R@3	MRR	MAP
FP16	—	—	—	—	—	—	—
NF4	—	—	—	—	—	—	—
GPTQ	—	—	—	—	—	—	—
AWQ	—	—	—	—	—	—	—
HQQ	—	—	—	—	—	—	—

Table 22: RAG efficiency metrics for quantization methods

Method	Retrieval Time (ms)	RAG Gen Time (ms)	No-RAG Gen Time (ms)	RAG Throughput (tok/s)	No-RAG Throughput (tok/s)	Generation Speedup
FP16	—	—	—	—	—	—
NF4	—	—	—	—	—	—
GPTQ	—	—	—	—	—	—
AWQ	—	—	—	—	—	—
HQQ	—	—	—	—	—	—