Comprehensive Evaluation of Quantization Methods for Edge LLM Deployment with RAG Focus

Zahraa Selim, Menna Hamed, Wesam Ahmed,
Sohyla Said, Sara Basheer, 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`

December 10, 2025

1 Methodology

1.1 Overview

This study conducts a comprehensive evaluation of post-training quantization techniques applied to the Mistral-7B model, with particular focus on Retrieval-Augmented Generation (RAG) capabilities. Our experimental protocol follows a two-phase approach: (1) quantize the base model on high-memory hardware, and (2) evaluate compressed models on resource-constrained edge hardware to simulate realistic deployment scenarios. All evaluations assess three dimensions: computational efficiency, general task performance, and RAG-specific capabilities.

1.2 Hardware and Software Environment

1.2.1 Quantization Environment

Model quantization is performed on cloud-based GPU instances with sufficient memory to load and process the full-precision model:

- **Platform:** Kaggle
- **GPU:** NVIDIA Tesla P100 (16GB VRAM)

The P100 provides sufficient memory bandwidth and compute capacity for the calibration-based quantization algorithms (GPTQ, AWQ), which require multiple forward passes through calibration datasets to compute optimal quantization parameters. This separation of quantization and evaluation environments reflects realistic deployment workflows where model compression is performed once on powerful hardware before deployment to edge devices.

1.2.2 Evaluation Environment

All performance evaluations are conducted on edge-class hardware to assess real-world deployment viability:

- **Platforms:** Kaggle and Google Colab free-tier instances

- **GPU:** NVIDIA Tesla T4 (16GB VRAM)

The Tesla T4 GPU represents a common edge deployment target, offering a balance between performance and power efficiency (70W TDP). Its 16GB memory constraint necessitates model compression for serving 7B parameter models, making it an ideal testbed for evaluating quantization effectiveness.

1.3 Base Model

We select **Mistral-7B-v0.1** [?] as our baseline model for the following reasons:

- **Architecture:** Modern transformer with Grouped Query Attention (GQA) and Sliding Window Attention (SWA), representative of current LLM designs
- **Scale:** 7.24 billion parameters—large enough to exhibit quantization challenges while remaining evaluable on edge hardware
- **Performance:** Strong baseline capabilities across reasoning, knowledge, and generation tasks
- **Format:** FP16 baseline requiring 13.49 GB storage, necessitating compression for edge deployment

The FP16 baseline model serves as our reference point for all performance comparisons, with measurements conducted under identical evaluation protocols to ensure fair comparison.

1.4 Compression Techniques

We evaluate four state-of-the-art post-training quantization (PTQ) methods, all targeting 4-bit precision to achieve approximately 3.6× compression:

1.4.1 NF4 (NormalFloat 4-bit)

NF4 employs an information-theoretically optimal quantization data type designed for normally distributed weights [?]. The method leverages the empirical observation that pre-trained neural network weights follow approximately Gaussian distributions.

Technical Approach:

NF4 constructs a 4-bit data type by quantizing weight distributions into 16 bins positioned at the quantiles of a standard normal distribution $\mathcal{N}(0, 1)$. For a weight w drawn from $\mathcal{N}(0, \sigma^2)$, the quantization maps w to the nearest quantile bin:

$$\hat{w} = \text{NF4}(w) = \sigma \cdot q_i, \quad \text{where } i = \arg\min_j |w - \sigma \cdot q_j| \tag{1}$$

where q_i are the 16 quantile values: $\{-1.0, -0.6962, -0.5251, ..., 0.5251, 0.6962, 1.0\}$.

Double Quantization:

To further reduce memory overhead, NF4 quantizes the quantization constants (scales σ) themselves using blockwise FP8 quantization. For a tensor divided into blocks of size 64, each block's scale is stored in 8-bit floating point rather than FP32, reducing quantization metadata by 75%.

Configuration:

- Quantization type: NF4 (16 quantile bins)
- Double quantization: Enabled (FP8 scales)
- Compute dtype: FP16 (dequantized computation)
- Block size: 64 (default bitsandbytes configuration)

- On-the-fly quantization: Model quantized during loading

Implementation:

Listing 1: NF4 Quantization Configuration

```
from transformers import BitsAndBytesConfig, AutoModelForCausalLM
import torch

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

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

1.4.2 GPTQ (Generative Pre-trained Transformer Quantization)

GPTQ is a layer-wise quantization algorithm that minimizes reconstruction error using second-order information [?]. The method reformulates optimal weight quantization as a series of layerwise least-squares problems.

Mathematical Formulation:

For each layer's weight matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, GPTQ solves:

$$\min_{\hat{\mathbf{W}}} \|\mathbf{W}\mathbf{X} - \hat{\mathbf{W}}\mathbf{X}\|_F^2 \tag{2}$$

where $\mathbf{X} \in \mathbb{R}^{d_{\text{in}} \times n}$ represents n calibration samples, and $\hat{\mathbf{W}}$ is the quantized weight matrix.

GPTQ employs a greedy coordinate descent strategy inspired by Optimal Brain Quantization (OBQ) [?], quantizing weights column-by-column while compensating for quantization errors in subsequent weights. The key innovation is using the Hessian matrix $\mathbf{H} = \frac{2}{n}\mathbf{X}\mathbf{X}^T$ to guide error compensation:

$$w_{q'} \leftarrow w_{q'} - \delta_q \cdot \frac{H_{qq'}}{H_{qq}} \quad \forall q' > q \tag{3}$$

where $\delta_q = w_q - \hat{w}_q$ is the quantization error for weight w_q.

Group-wise Quantization:

GPTQ applies quantization in groups to balance precision and compression. Each group of $g = 128$ consecutive weights shares quantization parameters (scale s and zero-point z):

$$\hat{w}_i = \left\lfloor \frac{w_i - z \cdot s}{s} \right\rceil, \quad s = \frac{\max(w_g) - \min(w_g)}{2^b - 1} \tag{4}$$

where $b = 4$ bits and $\lfloor \cdot \rceil$ denotes rounding to nearest integer.

Quantization Process:

We quantized Mistral-7B-v0.1 using AutoGPTQ on Tesla P100 GPUs:

- **Base model:** mistralai/Mistral-7B-v0.1 (7.24B parameters)
- **Precision:** 4-bit integer quantization

- **Group size:** 128 (weights per quantization group)
- **Calibration dataset:** [To be specified]
- **Calibration samples:** 128 sequences of maximum length 512 tokens
- **Damping factor:** 0.01 (Hessian regularization: $\mathbf{H} + \lambda\mathbf{I}$)
- **Memory optimization:** CPU offloading enabled (cache_examples_on_gpu=False)
- **Quantization time:** 22 minutes on P100 GPU

Implementation:

Listing 2: GPTQ Quantization Process

```python
from auto_gptq import AutoGPTQForCausalLM, BaseQuantizeConfig
from transformers import AutoTokenizer

# Configure quantization
quant_config = BaseQuantizeConfig(
    bits=4,
    group_size=128,
    desc_act=False,
    damp_percent=0.01
)

# Load model
model = AutoGPTQForCausalLM.from_pretrained(
    "mistralai/Mistral-7B-v0.1",
    quantize_config=quant_config,
    max_memory={0: "10GiB", "cpu": "50GiB"}
)

# Quantize with calibration data
model.quantize(
    calibration_data,
    use_triton=False,
    batch_size=1,
    cache_examples_on_gpu=False
)

# Save quantized model
model.save_quantized(
    "./mistral-7b-gptq-4bit",
    use_safetensors=True
)
```

Optimized Inference:

For evaluation, we employ ExLlamaV2, an optimized inference engine providing 2-3× speedup through specialized CUDA kernels for GPTQ models. ExLlamaV2 optimizations include custom kernels for 4-bit matrix multiplication, fused attention and MLP operations, and lazy KV cache loading.

1.4.3 AWQ (Activation-aware Weight Quantization)

AWQ protects salient weights based on activation magnitudes, preserving model accuracy through selective mixed-precision quantization [?]. The method identifies and protects the top 1% most influential weights while aggressively quantizing the remaining 99%.

Salient Weight Identification:

AWQ computes per-channel weight importance based on activation statistics. For weight matrix $\mathbf{W}$ and activation matrix $\mathbf{X}$, the importance of channel c is:

$$s_c = \|\mathbf{W}_{:,c}\|_2 \cdot \|\mathbf{X}_{c,:}\|_2 \tag{5}$$

Channels with highest s_c values are retained at higher precision (FP16) while others are quantized to 4-bit.

Per-channel Scaling:

AWQ applies learned per-channel scaling factors to balance activation magnitudes before quantization:

$$\mathbf{W}' = \mathbf{W} \cdot \text{diag}(\mathbf{s}), \quad \mathbf{X}' = \text{diag}(\mathbf{s}^{-1}) \cdot \mathbf{X} \tag{6}$$

where $\mathbf{s}$ are optimized to minimize quantization error while maintaining $\mathbf{W}'\mathbf{X}' = \mathbf{W}\mathbf{X}$.

Configuration:

- **Precision:** 4-bit integer weights, FP16 salient weights
- **Group size:** 128
- **Zero point:** Enabled (asymmetric quantization)
- **Version:** GEMM (optimized matrix multiplication kernels)
- **Salient weight ratio:** 1% (top channels protected)

Implementation:

We use pre-quantized AWQ models from TheBloke's repository to avoid redundant quantization compute:

Listing 3: AWQ Model Loading

```python
from awq import AutoAWQForCausalLM

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

1.4.4 HQQ (Half-Quadratic Quantization)

HQQ performs fast, calibration-free quantization by optimizing a custom half-quadratic loss function [?]. Unlike GPTQ and AWQ, HQQ requires no calibration data, making it suitable for rapid quantization without dataset access.

Half-Quadratic Optimization:

HQQ formulates quantization as minimizing:

$$\mathcal{L}(\mathbf{Q}, \mathbf{s}, \mathbf{z}) = \|\mathbf{W} - \mathbf{s} \odot \mathbf{Q} - \mathbf{z}\|_F^2 \tag{7}$$

where $\mathbf{Q}$ are quantized values, $\mathbf{s}$ are scales, $\mathbf{z}$ are zero-points, and $\odot$ denotes element-wise multiplication. The loss is optimized iteratively using a half-quadratic splitting technique that alternates between updating $\mathbf{Q}$ (quantization) and $(\mathbf{s}, \mathbf{z})$ (calibration parameters).

Row-wise Quantization:

HQQ applies quantization along weight matrix rows (axis=1), computing separate scales and zero-points for each output channel:

$$\hat{\mathbf{W}}_{i,:} = \text{quant}(\mathbf{W}_{i,:}, s_i, z_i) \quad \forall i \in [1, d_{\text{out}}] \tag{8}$$

This per-channel quantization preserves output distribution characteristics while avoiding the computational cost of Hessian-based methods.

Configuration:

- **Precision:** 4-bit integer
- **Group size:** 64
- **Quantization axis:** 1 (row-wise)
- **Compute dtype:** FP16
- **Optimization:** On-the-fly during model loading
- **Calibration:** None required

Implementation:

Listing 4: HQQ On-the-fly Quantization

```python
from hqq.models.hf.base import AutoHQQHFModel
from hqq.core.quantize import BaseQuantizeConfig
import torch

# Configure quantization
quant_config = BaseQuantizeConfig(
    nbits=4,
    group_size=64,
    axis=1  # Row-wise quantization
)

# Load and quantize model
model = AutoHQQHFModel.from_pretrained(
    "mistralai/Mistral-7B-v0.1",
    torch_dtype=torch.float16
)

model.quantize_model(
    quant_config=quant_config,
    device='cuda'
)
```

1.4.5 Quantization Summary

Table 1: Quantization methods comparison

Method	Calibration Required	Group Size	Optimization Basis	Quantization Time
NF4	No	64	Information theory	On-the-fly
GPTQ	Yes	128	Hessian (2nd-order)	20-25 min
AWQ	Yes	128	Activation-aware	50-60 min
HQQ	No	64	Half-quadratic	On-the-fly

All methods target 4-bit precision, achieving theoretical compression ratios of 3.6-4.0× relative to FP16 baseline. NF4 and HQQ quantize on-the-fly during model loading, while GPTQ requires offline quantization on P100 GPUs. AWQ uses pre-quantized checkpoints from TheBloke's repository for efficiency.

1.5 Calibration Dataset

For methods requiring calibration data (GPTQ and AWQ), we develop a custom RAG-specific calibration dataset that reflects realistic retrieval-augmented generation workloads. Unlike standard calibration approaches that use generic text corpora (e.g., C4, WikiText), our dataset is explicitly structured to match RAG prompt formats and complexity distributions.

1.5.1 Dataset Construction

We generate 128 calibration samples following the structured RAG format:

```
[QUERY]: {question}
[RETRIEVED DOCUMENTS]:
Document 1: {context_1}
Document 2: {context_2}
...
[ANSWER]: {answer}
```

This format mirrors production RAG systems where queries are paired with retrieved documents and expected answers, ensuring quantization parameters are optimized for the actual inference distribution.

1.5.2 Source Datasets

Calibration samples are synthesized from three complementary sources:

- **SQuAD v2** [?]: Provides high-quality question-answer pairs with clean, focused contexts for short-context samples
- **HotpotQA** [?]: Supplies multi-hop reasoning examples requiring information synthesis across multiple documents
- **Wikitext-103** [?]: Contributes general-domain passages as distractor documents and context padding

1.5.3 Distribution Strategy

We employ a stratified sampling approach targeting four complexity tiers that represent the operational envelope of RAG systems:

Table 2: RAG calibration dataset distribution

Category	**Samples**	**Percentage**	**Token Range**	**Characteristics**
Short-context	32	25%	256–512	Single document, direct extraction
Medium-context	40	31%	1024–2048	2–3 documents, comparison
Long-context	32	25%	3072–4096	5–7 documents, synthesis
Multi-hop	24	19%	Variable	8–10 documents, complex reasoning
Total	**128**	**100%**	**256–4096**	—

Short-context Samples (32 samples, 256–512 tokens) These samples simulate straightforward retrieval scenarios where a single relevant document contains the answer. We extract question-context-answer triples directly from SQuAD v2, representing the baseline RAG use case of locating information in focused passages.

Medium-context Samples (40 samples, 1024–2048 tokens) Medium-context samples introduce multi-document scenarios requiring comparison or selective attention. For each sample, we combine one relevant SQuAD context with 1–2 distractor passages from Wikitext-103, forcing the model to identify and focus on pertinent information among noise.

Long-context Samples (32 samples, 3072–4096 tokens) Long-context samples test the model's ability to maintain coherence and extract information from extended contexts. We construct these by:

1. Selecting multi-hop questions from HotpotQA requiring reasoning across multiple facts
2. Combining 5–7 supporting context passages from HotpotQA and Wikitext-103
3. Shuffling document order to prevent positional bias
4. Padding to target length range with additional Wikipedia passages

Multi-hop Samples (24 samples, 8–10 documents) The most challenging category simulates complex retrieval scenarios requiring multi-step reasoning. We filter HotpotQA for comparison and bridge questions (requiring connecting information across documents), then construct contexts with 8–10 documents including both supporting and distractor passages. These samples stress-test the quantized model's capacity for long-range dependencies and complex inference chains.

1.5.4 Dataset Statistics

The resulting calibration dataset exhibits the following statistical properties:

- **Total samples:** 128
- **Mean length:** 1,847 tokens (word count approximation)
- **Median length:** 1,523 tokens
- **Range:** 256–4,096 tokens
- **Standard deviation:** 1,124 tokens

Length distribution across bins:

- <512 tokens: 32 samples (25.0%)
- 512–1K tokens: 15 samples (11.7%)
- 1K–2K tokens: 41 samples (32.0%)
- 2K–4K tokens: 32 samples (25.0%)
- >4K tokens: 8 samples (6.3%)

This distribution ensures quantization algorithms observe representative activation patterns across the full spectrum of RAG workloads, from simple single-document retrieval to complex multi-hop reasoning over extensive contexts.

1.5.5 Calibration Protocol

For GPTQ quantization, we apply the following protocol:

- **Samples:** All 128 calibration samples
- **Sequence length:** Maximum 2048 tokens (truncated if necessary)
- **Batch size:** 1 (due to variable lengths)
- **Processing:** Sequential, no example caching on GPU

- **Hessian computation:** Accumulated across all calibration forward passes

For AWQ quantization, we use the same 128 samples with:

- **Sequence length:** Maximum 2048 tokens
- **Activation collection:** Per-channel statistics aggregated across all samples
- **Salient weight identification:** Top 1% channels protected based on activation magnitudes

By tailoring the calibration dataset to RAG-specific patterns, we ensure quantization algorithms preserve the activation distributions and weight sensitivities most critical for retrieval-augmented generation performance.

1.6 RAG Pipeline Configuration

1.6.1 Pipeline Architecture

Our RAG (Retrieval-Augmented Generation) pipeline implements a modular architecture consisting of five core components that work in sequence to enable context-aware question answering:

1. **Document Processing:** Extract and clean text from source documents (PDF, TXT, Markdown)
2. **Text Chunking:** Split documents into semantically coherent segments
3. **Embedding:** Convert text chunks into dense vector representations
4. **Vector Indexing:** Store and index embeddings for efficient similarity search
5. **Retrieval & Generation:** Query the index and generate contextualized answers

The pipeline is implemented as a reusable framework that can be applied to any of our compressed models, enabling direct comparison of RAG performance across quantization methods.

1.6.2 Document Processing

Text Extraction:

- **PDF Processing:** PyPDF2-based extraction with page-level granularity
- **Text Normalization:** Whitespace normalization, OCR error correction, quote standardization
- **Cleaning Operations:**
  - Remove page numbers and headers
  - Strip citation references ([1], (Author, 2020))
  - Remove URLs and excessive whitespace
  - Fix common ligature errors (fi, fl)

Table 3: Document processing parameters

Parameter	Default Value	Description
remove_headers	true	Strip page headers and numbers
remove_citations	true	Remove citation markers
extract_sections	false	Parse document sections

1.6.3 Text Chunking Strategy

We employ a **semantic chunking** strategy that respects natural document boundaries while maintaining optimal chunk sizes for embedding and retrieval. This approach outperforms fixed-size chunking by preserving contextual coherence.

Chunking Algorithm:

- **Strategy:** Semantic (paragraph-aware)
- **Primary Delimiter:** Double newlines (paragraph boundaries)
- **Fallback:** Sentence-level tokenization using NLTK punkt tokenizer
- **Overlap Mechanism:** Sliding window with configurable overlap to maintain context continuity across chunk boundaries

Table 4: Text chunking parameters

Parameter	Default Value	Description
strategy	semantic	Chunking strategy (semantic, sentence, fixed)
chunk_size	512	Target chunk size in tokens
chunk_overlap	50	Overlap between consecutive chunks
min_chunk_size	100	Minimum viable chunk size

Chunk Metadata: Each chunk includes metadata for traceability and filtering:

- **chunk_id:** Unique identifier (chunk_0, chunk_1, ...)
- **page_number:** Source page in original document
- **start_char / end_char:** Character offsets in source
- **tokens:** Word count for the chunk
- **section:** Optional section header (if extracted)

1.6.4 Embedding Model

We use **sentence-transformers/all-MiniLM-L6-v2** as our embedding model, balancing quality and efficiency for retrieval tasks.

Model Characteristics:

- **Architecture:** Distilled from Microsoft MiniLM
- **Embedding Dimension:** 384
- **Max Sequence Length:** 256 tokens
- **Training:** Fine-tuned on 1B+ sentence pairs
- **Performance:** SBERT benchmark score: 68.06 (semantic similarity)

Table 5: Embedding model parameters

Parameter	Default Value	Description
model_name	all-MiniLM-L6-v2	SentenceTransformer model identifier
batch_size	32	Batch size for embedding generation
normalize	true	L2 normalization of embeddings
device	cuda	Compute device (cuda, mps, cpu)

Implementation:

Listing 5: Embedding Generation

```python
from sentence_transformers import SentenceTransformer

model = SentenceTransformer(
    'sentence-transformers/all-MiniLM-L6-v2',
    device='cuda'
)

embeddings = model.encode(
    texts,
    batch_size=32,
    show_progress_bar=True,
    normalize_embeddings=True,
    convert_to_numpy=True
)
```

1.6.5 Vector Store and Indexing

We use **ChromaDB** as our vector database, providing efficient similarity search with multiple distance metrics.

Vector Store Configuration:

- **Database:** ChromaDB (persistent or in-memory)
- **Index Type:** HNSW (Hierarchical Navigable Small World)
- **Distance Metric:** Cosine similarity (default)
- **Storage:** Optional persistent storage for index reuse

Table 6: Vector store parameters

Parameter	Default Value	Description
collection_name	rag_documents	Index collection identifier
persist_directory	null	Directory for persistent storage (null=in-memory)
distance_metric	cosine	Distance function (cosine, l2, ip)

Distance-to-Similarity Conversion: ChromaDB returns distances that must be converted to similarity scores [0, 1]:

- **Cosine:** similarity $= 1 - \frac{d^2}{2}$ where d is L2 distance of normalized vectors
- **L2:** similarity $= \frac{1}{1+d}$ (exponential decay)
- **Inner Product:** similarity $= \frac{d+2}{2}$ (normalized to [0, 1])

1.6.6 Retrieval Strategy

Our retrieval system implements advanced techniques beyond simple nearest-neighbor search:

Base Retrieval:

- **Top-K Selection:** Retrieve top-3 most similar chunks by default
- **Similarity Threshold:** Configurable minimum similarity score (default: 0.0)
- **Metadata Filtering:** Optional filtering by page number, section, etc.

Re-ranking (Optional): Hybrid retrieval combining semantic and lexical matching:

- **Semantic Score:** Original embedding similarity (70% weight)

- **Lexical Score:** Token overlap between query and chunk (30% weight)
- **Formula:** rerank_score $= 0.7 \times$ cosine_sim $+ 0.3 \times$ token_overlap

Diversity Mechanism (Optional): Maximal Marginal Relevance (MMR) to reduce redundancy:

- **Objective:** Balance relevance and diversity in retrieved chunks
- **Formula:** $\mathrm{MMR} = \lambda \times \mathrm{Sim}(q, c) - (1 - \lambda) \times \max[\mathrm{Sim}(c, S)]$
- where q is query, c is candidate chunk, S is selected chunks, λ is diversity parameter

Table 7: Retrieval parameters

Parameter	**Default Value**	**Description**
top_k	3	Number of chunks to retrieve
similarity_threshold	0.0	Minimum similarity score
rerank	false	Enable hybrid re-ranking
diversity_penalty	0.0	MMR diversity parameter [0, 1]

1.6.7 Answer Generation

The generation component uses the compressed LLM with retrieved context to produce grounded answers.

Prompt Engineering: We design prompts to encourage faithful, concise answers based on retrieved context:

Listing 6: RAG Generation Prompt Template

```
prompt = f"""Use the following context to answer the question.
Provide a clear, direct answer based on the information given.

Context:
{retrieved_context}

Question: {user_query}

Answer:"""
```

Generation Parameters: Carefully tuned to balance faithfulness and naturalness:

- **Max New Tokens:** 128 (concise answers)
- **Temperature:** 0.3 (low for factual accuracy)
- **Top-p:** 0.9 (nucleus sampling)
- **Repetition Penalty:** 1.15 (prevent loops)
- **Sampling:** Enabled (allows natural phrasing)

Table 8: Answer generation parameters

Parameter	Default Value	Description
max_new_tokens	128	Maximum answer length
temperature	0.3	Sampling temperature
top_p	0.9	Nucleus sampling threshold
do_sample	true	Enable sampling vs greedy
repetition_penalty	1.15	Penalty for repeated tokens
use_chat_template	true	Use model's chat template if available

Answer Validation: Post-processing to ensure quality:

- **Truncation:** Limit to 4 sentences maximum
- **Context Truncation:** Cap context at 2000 characters to prevent overwhelming
- **Retry Mechanism:** If answer appears problematic (too short, repetitive, verbatim copying), retry with simplified prompt and lower temperature (0.2)
- **Fallback:** Return "The information is not provided in the given context" for empty/invalid responses

1.6.8 RAG Evaluation Dataset

We create a custom technical QA dataset from documentation corpora to evaluate RAG performance:

Dataset Characteristics:

- **Domain:** Technical documentation (ML frameworks, APIs)
- **Size:** 10-50 question-answer pairs per evaluation
- **Question Types:**
  - Factual: "What is the default learning rate?"
  - Procedural: "How do you initialize a model?"
  - Comparative: "What's the difference between X and Y?"
- **Answer Format:** Short-form answers (1-3 sentences)
- **Ground Truth:** Human-verified reference answers

Table 9: RAG evaluation dataset parameters

Parameter	Default Value	Description
num_questions	10	Number of QA pairs to evaluate
dataset_path	null	Path to custom QA JSON file
compare_no_rag	true	Evaluate without retrieval baseline
save_detailed_responses	false	Save individual responses to file

Evaluation Protocol:

1. Index source documents using the RAG pipeline
2. For each test question:
   - Retrieve top-K relevant chunks
   - Generate answer with context (RAG)

    - Generate answer without context (no-RAG baseline)

3. Compute metrics comparing predictions to reference answers
4. Aggregate results across all questions

1.6.9 Pipeline Integration with Compressed Models

The RAG pipeline is model-agnostic and interfaces with any compressed model through a unified ModelInterface:

Listing 7: RAG Pipeline Initialization

```python
from rag import RAGPipeline

# Load compressed model
model_interface = load_compressed_model("NF4")

# Initialize RAG pipeline
rag_config = {
    "chunking": {"strategy": "semantic", "chunk_size": 512},
    "embedding": {"model_name": "all-MiniLM-L6-v2"},
    "retrieval": {"top_k": 3, "rerank": False},
    "generation": {"temperature": 0.3, "max_new_tokens": 128}
}

pipeline = RAGPipeline(rag_config)
pipeline.setup(model_interface)

# Index documents
pipeline.index_documents("technical_docs.pdf")

# Evaluate
results = pipeline.evaluate(test_questions, compare_no_rag=True)
```

This design enables fair comparison of RAG performance across all compression methods, as all components except the generation model remain constant.

1.7 Evaluation Metrics and Benchmarks

We assess compressed models across three evaluation dimensions: computational efficiency, general task performance, and RAG-specific capabilities. All evaluations are conducted on the same Tesla T4 GPU to ensure fair comparison.

1.7.1 Computational Efficiency Metrics

Efficiency metrics quantify the speed, memory, and energy improvements achieved through quantization.

Table 10: Time performance metrics

Metric	Description
Latency (ms/token)	End-to-end generation speed measured as mean time per generated token across 10 inference runs with 3 warmup iterations
Time-to-First-Token (TTFT, ms)	Initial response latency measuring time from prompt submission to first token generation
Prefill Latency (ms)	Prompt processing time in parallel
Decode Latency (ms/token)	Autoregressive token generation time
Throughput (tokens/s)	Sustained generation rate: Throughput = $\frac{\sum_{i=1}^{10} \text{tokens}_i}{\sum_{i=1}^{10} \text{time}_i}$

Time Performance Metrics

Table 11: Space performance metrics

Metric	**Description**
Model Size (GB)	Disk storage requirements: $\text{Size}_{\text{GB}} = \frac{\sum(\text{element\_size} \times \text{numel})}{1024^3}$
Peak Memory (MB)	Maximum GPU memory allocated during inference
Bits per Parameter	Average quantization level: $\text{Bits/param} = \frac{\text{Size}_{\text{bytes}} \times 8}{\text{Total parameters}}$
Compression Ratio	Size reduction relative to FP16: $\text{Compression} = \frac{\text{Size}_{\text{FP16}}}{\text{Size}_{\text{quantized}}}$
Memory Efficiency	Ratio of model size to peak memory: $\text{Efficiency} = \frac{\text{Model Size (GB)}}{\text{Peak Memory (GB)}}$
KV Cache Size (MB)	Attention cache memory for sequence length 2048: $\text{KV Size} = 2 \times L \times B \times H \times S \times D \times \text{bytes}$

Space Performance Metrics

1.7.2 General Task Performance Benchmarks

We evaluate compressed models on standard language modeling and reasoning benchmarks to assess general capability preservation.

Language Modeling: Perplexity Perplexity (PPL) measures next-token prediction quality on WikiText-2 test set:

$$\text{PPL}(W) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i|w_{<i})\right) \tag{9}$$

Table 12: Perplexity evaluation parameters

Parameter	**Value**
Dataset	WikiText-2 test split
Samples	100 passages
Max length	512 tokens
Stride	512 (sliding window)
Min text length	100 characters

Core Task Benchmarks We evaluate models using the Language Model Evaluation Harness with standardized protocols. Our benchmark selection is organized into four categories:

Baseline Reasoning (0-shot):

- **HellaSwag**: Sentence completion requiring commonsense reasoning. Tests intuitive understanding preservation. Metric: normalized accuracy.
- **Winogrande**: Pronoun disambiguation requiring commonsense knowledge. Evaluates semantic understanding. Metric: accuracy.
- **PIQA**: Physical commonsense reasoning. Tests intuitive physics knowledge. Metric: normalized accuracy.

- **ARC-Easy**: Grade-school science questions. Assesses factual knowledge retention. Metric: normalized accuracy.
- **ARC-Challenge**: Challenge-level science questions. Tests complex multi-hop reasoning. Metric: normalized accuracy.

Core RAG Tasks (0-shot):

- **SQuAD**: Extractive question answering from Wikipedia paragraphs (1-2 paragraphs). Tests accurate answer extraction from short contexts. Metric: F1 score and exact match.
- **TriviaQA**: Open-domain QA with evidence documents. Tests robustness to longer, noisier contexts. Metric: exact match.

RAG with Complex Reasoning (3-shot):

- **DROP**: Discrete reasoning over paragraphs requiring arithmetic, counting, sorting. Tests numerical reasoning preservation. Metric: F1 score.
- **RACE**: Reading comprehension from educational exams requiring inference beyond surface matching. Tests complex comprehension. Metric: accuracy.

Fact Verification (0-shot):

- **BoolQ**: Yes/no question answering requiring statement verification against passages. Tests binary classification for hallucination detection. Metric: accuracy.

Table 13: Core task benchmark organization

Category	**Tasks**	**Few-shot**	**Est. Time (min)**	**RAG Relevance**
Baseline Reasoning	5	0	15	Establishes non-RAG baseline
Core RAG	2	0	20	Direct context QA
RAG + Reasoning	2	3	12	Complex multi-step inference
Verification	1	0	5	Hallucination detection
Total	**10**	—	**52**	—

All evaluations use greedy decoding (temperature=0) for reproducibility. Estimated times are per quantization method on Tesla T4 GPU.

1.7.3 RAG-Specific Performance Metrics

We develop custom evaluations to assess quantization impact on RAG-critical capabilities: attention preservation, context handling, and retrieval-augmented generation quality.

Attention Preservation This metric evaluates whether quantized models maintain proper attention to relevant documents in multi-document contexts.

Table 14: Attention preservation evaluation setup

Parameter	**Value**
Dataset	Natural Questions validation (500 samples)
Documents per query	5 (1 relevant + 4 distractors)
Document source	Wikipedia passages ($\leq$ 250 chars)
Max context length	512 tokens
Generation length	20 tokens
Target samples	50 successful evaluations

Attention Extraction Protocol:

We extract document-level attention from model outputs during generation by:

1. Capturing attention weights from last transformer layer
2. Averaging across attention heads: $[B, H, S, S] \to [S, S]$
3. Extracting attention from last generated token to input: $[S, S] \to [S]$
4. Aggregating token-level attention to document level by equal division
5. Normalizing to probability distribution: $\sum_{i=1}^{5} a_i = 1.0$

Attention snapshots are captured at three generation stages: start (token 1), middle (token $\lfloor N/2 \rfloor$), and end (token N), where N is the number of generated tokens.

Metrics:

Preservation Metrics:

- **Precision@1:** Fraction of queries where model assigns highest attention to relevant document. Range: [0, 1], higher is better.
- **Mean/Median Rank:** Average rank of relevant document by attention weight. Range: [1, 5], lower is better (1 = top-ranked).
- **Mean Attention on Answer:** Average attention weight on relevant document. Range: [0, 1], higher indicates stronger focus.
- **Attention Concentration (Gini):** Distribution uniformity measured via Gini coefficient. Higher values indicate focused attention on fewer documents.

Attention Drift Metrics:

- **Mean Total Drift:** Average L1 distance between initial and final attention distributions:

$$\text{Drift} = \|\mathbf{a}_{\text{final}} - \mathbf{a}_{\text{initial}}\|_1 \tag{10}$$

  where $\mathbf{a} \in \mathbb{R}^5$ is the document-level attention distribution. Range: [0, 2], lower indicates more stable attention.
- **Mean Answer Drift:** Attention change on relevant document specifically:

$$\text{Answer Drift} = |a_{\text{relevant}}^{\text{final}} - a_{\text{relevant}}^{\text{initial}}| \tag{11}$$

  Lower values indicate stable focus on relevant information during generation.
- **Mean Max Drift:** Maximum single-document attention change across all documents, indicating which document experienced largest attention shift.
- **Answer Drift Direction:** Signed difference indicating whether attention to answer document increased (+) or decreased (-) during generation.

Quality Correlation:

- **Exact Match:** Fraction of answers containing ground truth string (case-insensitive).
- **Mean F1:** Token-level F1 score between generated answers and ground truth, measuring whether attention preservation correlates with answer quality.

Context Length Degradation This metric evaluates performance degradation as context length increases, simulating long-document RAG scenarios.

Table 15: Context length degradation evaluation setup

Parameter	**Value**
Dataset	SQuAD v2 validation (300 samples)
Context lengths tested	[512, 1024, 2048, 4096] tokens
Filler passages	WikiText-103
Samples per length	25
Answer position	Middle of context (surrounded by filler)
Generation length	20 tokens

Context Assembly Protocol:

For each evaluation sample:

1. Start with question + answer-containing context (base passage from SQuAD)
2. Calculate token count of base passage
3. Add random Wikipedia filler passages until reaching target length
4. Shuffle filler to surround answer passage (placing it at target position)
5. Truncate to exact target length if exceeded
6. Append "Answer:" prompt

This ensures the answer is embedded in realistic long-context scenarios rather than isolated at the end, testing the model's ability to locate and extract information from extended contexts with distractors.

Metrics:

Accuracy Metrics:

- **F1 by Length:** Token-level F1 score at each context length, measuring answer quality degradation
- **Accuracy by Length:** Exact match accuracy at each context length (binary correctness)

Degradation Analysis:

- **Slope per 1K tokens:** Linear regression slope of F1 vs. context length:

$$F1 = \beta_0 + \beta_1 \times \text{context\_length} + \epsilon \qquad (12)$$

  Negative slopes indicate degradation, with steeper slopes showing worse robustness to long contexts. Interpretation:

  - $|\beta_1| < 0.001$: negligible degradation
  - $0.001 \leq |\beta_1| < 0.01$: minimal degradation

  - $0.01 \leq |\beta_1| < 0.05$: moderate degradation
  - $|\beta_1| \geq 0.05$: significant degradation

- **R-squared:** Regression goodness-of-fit, indicating whether degradation follows linear pattern. Values near 1.0 suggest consistent linear degradation; lower values indicate non-linear or irregular patterns.

Position Analysis:

- **Mean F1 by Position:** Average performance when answer is at start, middle, or end of context
- **F1 Standard Deviation:** Variability across different context lengths
- **Min/Max F1:** Performance range indicating robustness

RAG System Evaluation Complete evaluation of retrieval-augmented generation combining retrieval quality, answer generation quality, and system efficiency.

Table 16: RAG system evaluation setup

Parameter	Value
Test questions	10-50 Q&A pairs
Documents indexed	Technical documentation corpus
Retrieval method	Dense vector search (sentence-transformers)
Top-k chunks	3 per query
Max generation tokens	50
Compare no-RAG baseline	Yes

Metrics:

Retrieval Quality:

Table 17: Retrieval quality metrics

Metric	Description
Context Sufficiency	Fraction of queries where retrieved contexts contain answer (80% token overlap threshold)
Context Precision	Query-context relevance via token overlap
Answer Coverage	Fraction of answer tokens present in retrieved contexts
Avg Retrieval Score	Mean similarity score from vector search
Retrieval Consistency	Standard deviation of retrieval scores across queries
Avg Context Length	Mean word count of retrieved context

Answer Quality:

Table 18: Answer generation quality metrics

Metric	**Description**
Exact Match (EM)	Binary correctness: perfect normalized string match
F1 Score	Token-level precision-recall harmonic mean
Faithfulness	Fraction of answer tokens present in retrieved context (groundedness)
ROUGE-1/2/L	N-gram overlap (unigram, bigram) and longest common subsequence (optional)
Avg Answer Length	Mean word count of generated answers

RAG vs. No-RAG Comparison:

- **F1 Gain:** Absolute improvement: $\Delta F1 = F1_{\text{RAG}} - F1_{\text{no-RAG}}$
- **F1 Gain Percent:** Relative improvement: $\frac{\Delta F1}{F1_{\text{no-RAG}}} \times 100\%$
- **EM Gain:** Exact match improvement with retrieval augmentation

Efficiency:

Table 19: RAG system efficiency metrics

Metric	**Description**
Avg Retrieval Time (ms)	Mean time to retrieve and rank top-k chunks
Avg RAG Generation Time (ms)	Mean time to generate answer with context
Avg No-RAG Generation Time (ms)	Baseline generation time without context
RAG Overhead (ms)	Additional latency: $(t_{\text{retrieval}} + t_{\text{RAG-gen}}) - t_{\text{no-RAG-gen}}$
RAG Overhead Percent	Relative overhead: $\frac{\text{overhead}}{t_{\text{no-RAG-gen}}} \times 100\%$

All RAG evaluations are conducted on the same Tesla T4 GPU with identical retrieval configurations to ensure fair comparison across quantization methods. Retrieval uses cached embeddings to isolate generation performance from embedding computation overhead.

2 Results

This section presents empirical results across three evaluation dimensions: computational efficiency (Section 2.1), task performance quality (Section 2.2), and RAG-specific capabilities (Section 2.3). All measurements are conducted on NVIDIA Tesla T4 (16GB VRAM) using Mistral-7B-v0.1 as the base model.

2.1 Computational Efficiency Analysis

2.1.1 Time Performance Metrics

Table 20: Latency measurements across quantization methods

Method	**Latency (ms/tok)**	**Latency Std**	**TTFT (ms)**	**TTFT Std**	**Prefill (ms)**	**Decode (ms/tok)**
FP16	62.72	0.19	69.47	0.23	69.47	62.72
NF4	81.12		177.16		187.97	79.09
GPTQ	1136.80		1113.00		1110.89	1117.76
AWQ	78.68		79.25		92.76	77.22
HQQ	661.65		664.73		663.17	649.84

Table 21: Throughput measurements across quantization methods

Method	**Throughput (tok/s)**	**Throughput Std**	**Total Tokens**	**Total Time (s)**	**Speedup vs FP16**
FP16	15.90	0.03	1280	80.52	1.00×
NF4	12.30				0.78×
GPTQ	0.88				0.06×
AWQ	12.79				0.81×
HQQ	1.51				0.10×

2.1.2 Space Performance Metrics

Table 22: Model size and compression metrics

Method	**Model Size (GB)**	**Total Parameters**	**Bits per Parameter**	**Compression Ratio**
FP16	13.49	7.24B	16.0	1.00×
NF4	3.74	7.24B	4.0	3.61×
GPTQ	3.87	7.24B	4.0	3.48×
AWQ	3.87	7.24B	4.0	3.49×
HQQ	3.74	7.24B	4.0	3.61×

Table 23: Memory utilization metrics

Method	Peak Memory (MB)	Memory Efficiency	Memory Reduction vs FP16
FP16	6906.26	2.00	1.00×
NF4	1859.11	2.06	3.77×
GPTQ	4415.90	0.90	1.59×
AWQ	1869.04	2.12	3.75×
HQQ	4721.1	0.81	1.49×

2.2 Task Performance Analysis

2.2.1 Language Modeling: Perplexity

Table 24: Perplexity on WikiText-2 test set

Method	Perplexity	Loss	Delta PPL vs FP16	Degradation (%)
FP16	8.80	2.174	0.00	0.0%
NF4	13.02		+0.23	+1.8%
GPTQ	12.85		+0.06	+0.5%
AWQ	13.47		+0.68	+5.3%
HQQ	13.5		+0.71	+5.6%

2.2.2 Core Task Benchmarks

Table 25: Baseline reasoning tasks

Method	HellaSwag	Winogrande	PIQA	ARC-Easy	ARC-Challenge
FP16	0.72			0.76	0.58
NF4	0.70			0.75	0.58
GPTQ	0.68			0.75	0.60
AWQ					
HQQ	0.69			0.72	0.5

Baseline Reasoning (0-shot)

Table 26: Core RAG tasks: reading comprehension

Method	SQuAD (F1)	TriviaQA (EM)
FP16		
NF4		
GPTQ		
AWQ		
HQQ		

Core RAG Tasks (0-shot)

Table 27: Complex reasoning over context

Method	DROP (F1)	RACE (Acc)
FP16		
NF4		
GPTQ		
AWQ		
HQQ		

RAG with Complex Reasoning (3-shot)

Table 28: Binary fact verification

Method	BoolQ (Acc)
FP16	
NF4	
GPTQ	
AWQ	
HQQ	

Fact Verification (0-shot)

2.3 RAG-Specific Performance Analysis

2.3.1 Attention Preservation

Table 29: Attention preservation metrics

Method	Precision @1	Mean Rank	Median Rank	Mean Attn on Answer	Gini Coeff	Samples
FP16	0.273	3.18	4.0	0.230	0.499	11
NF4						
GPTQ						
AWQ						
HQQ						

Table 30: Attention drift metrics

Method	Mean Total Drift	Mean Answer Drift	Mean Max Drift	Answer Drift Direction
FP16	0.402	0.093	0.192	0.005
NF4				
GPTQ				
AWQ				
HQQ				

Table 31: Attention preservation quality correlation

Method	Exact Match	Mean F1
FP16	0.364	0.039
NF4		
GPTQ		
AWQ		
HQQ		

2.3.2 Context Length Degradation

Table 32: Context length degradation analysis

Method	Slope per 1K tokens	R-squared	Degradation Interpretation	Samples per Length
FP16	-0.018	0.87	Mild linear drop	11
NF4				
GPTQ				
AWQ				
HQQ				

Table 33: Performance by context length

Method	F1 @ 512 tokens	F1 @ 1024 tokens	F1 @ 2048 tokens	F1 @ 4096 tokens	Acc @ 512 tokens
FP16	0.039	—	—	—	0.364
NF4					
GPTQ					
AWQ					
HQQ					

Table 34: Position analysis summary

Method	Mean F1 (middle pos)	Std F1	Min F1	Max F1
FP16	0.0393	0.112	0.0	0.38
NF4				
GPTQ				
AWQ				
HQQ				

2.3.3 RAG System Evaluation

Table 35: Retrieval quality metrics

Method	Context Sufficiency	Context Precision	Answer Coverage	Avg Score	Retrieval Consistency	Avg Context Length (words)
FP16	0.796	0.564	0.756	0.800	0.090	1308.5
NF4	0.796	0.564	0.756	0.800	0.090	1308.5
AWQ	0.796	0.564	0.756	0.800	0.090	1308.5
GPTQ						
HQQ						

Table 36: Answer generation quality

Method	Exact Match	F1 Score	F1 Std	Faithful-ness	Avg Answer Length (words)
FP16	0.0	0.217		0.559	33.75
NF4	0.0	0.205		0.539	31.95
AWQ	0.0	0.191		0.476	30.5
GPTQ					
HQQ					

Table 37: ROUGE scores (optional metrics)

Method	ROUGE-1	ROUGE-2	ROUGE-L
FP16	0.279	0.092	0.197
NF4	0.244	0.086	0.177
AWQ	0.243	0.088	0.181
GPTQ			
HQQ			

Table 38: RAG vs No-RAG comparison

Method	RAG F1	No-RAG F1	F1 Gain (abs)	F1 Gain (%)	RAG EM	No-RAG EM
FP16	0.217	0.190	+0.027	+14.2%	0.0	0.0
NF4	0.205	0.181	+0.024	+13.3%	0.0	0.0
AWQ	0.191	0.165	+0.025	+15.2%	0.0	0.0
GPTQ						
HQQ						

Table 39: RAG system efficiency

Method	Avg Retrieval Time (ms)	Retrieval Std (ms)	Avg RAG Gen Time (ms)	Avg No-RAG Gen Time (ms)	RAG Overhead (ms)	RAG Overhead (%)
FP16	24.7		8435.5	7855.2	+605.0	+7.7%
NF4	28.0		10443.9	9965.0	+506.9	+5.1%
AWQ	26.1		9993.9	7744.2	+2275.8	+29.4%
GPTQ						
HQQ						