Optimizing LLaMA 3.2 1B Using Quantization Techniques using Bitsandbytes for Efficient AI Deployment

Abstract

Large Language Models (LLMs) have transformed natural language processing, which has achieved state-of-the-art performance on various tasks. However, their high computational and memory requirements lead to significant challenges for deployment, especially on resource-constrained hardware. In this paper, we conduct a controlled experiment to optimize the LLaMA 3.2 1B model using post-training quantization techniques implemented using the 'Bitsandbytes' library. Evaluating multiple precision settings like BF16, FP16, INT8, and INT4 compare their accuracy, throughput, latency, and resource utilization tradeoffs. Experiments are conducted on a workstation GPU (NVIDIA T1000) for accuracy benchmarking and a cloud-based GPU (Nvidia T4 on Google Colab) for performance benchmarking. Our findings show that lower precision quantization can significantly reduce memory usage and improve throughput with minimal impact on model accuracy, providing valuable insights for efficient AI deployment for production environments.

Key words: Large Language Models (LLMs), Quantization Techniques, Post-Training Quantization (PTQ), Bitsandbytes, LLaMA 3.2 1B, Inference Efficiency

I. Introduction

Recent breakthroughs in large language models (LLMs) have redefined artificial intelligence by enabling systems to understand and produce text that closely resembles human language. Over the past decade, innovations such as GPT, BERT, and Meta's LLaMA have reshaped natural language processing, paving the way for more nuanced and context-sensitive interactions between humans and machines. Built on transformer architectures, these models have grown in both scale and intricacy, necessitating significant computational power for both training and real-time inference. However, as their capabilities expand, so do the challenges of deploying them—especially in settings with limited hardware resources.

A major obstacle in widespread LLM deployment is their high memory and computational demands, which make them unsuitable for consumer-grade GPUs or edge devices. To overcome these limitations, researchers have turned to various optimization methods, with quantization emerging as a particularly effective strategy. This process involves reducing the numerical precision of a model's weights and activations, thereby boosting execution efficiency and cutting down on memory usage. Although LLMs are typically trained using full 32-bit precision, the push for efficient deployment has spurred a transition to lower-precision formats such as BF16, FP16, INT8, and INT4. While BF16 and FP16 slightly lower precision with minimal impact on accuracy, the integer formats (INT8 and INT4) can drastically shrink model size and computational load, albeit sometimes at the cost of reduced accuracy.

Post-training quantization (PTQ) is a widely embraced technique that transforms a fully trained model into a lower-precision version without necessitating further retraining, making it highly practical for real-world applications. Tools like the 'Bitsandbytes' library have become indispensable, as they enable efficient conversion to FP16, INT8, and INT4 formats while largely preserving performance. Despite the extensive research on quantization, most studies have concentrated on large-scale models and high-end hardware, leaving an underexplored area regarding its impact on smaller models and resource-limited environments. Moreover, many investigations tend to focus on theoretical gains rather than robust empirical evaluations in practical scenarios.

This study addresses that gap by systematically examining the effects of post-training quantization on the LLaMA 3.2-1B model—a streamlined variant suitable for consumer GPUs. By applying FP16, INT8, and INT4 quantization, we analyze the trade-offs among model accuracy, inference speed, memory efficiency, and computational overhead. Accuracy tests were conducted using an NVIDIA T1000 GPU, while performance metrics such as throughput, latency, and GPU utilization were measured on a T4 GPU via Colab.

The results reveal that INT8 quantization offers an optimal balance between memory savings and accuracy, making it a strong candidate for practical applications. Although INT4 delivers superior memory reduction, its higher accuracy loss may restrict its use in precision-critical tasks. FP16, in comparison, provides modest computational improvements over BF16 in real-world inference settings. These findings offer valuable guidance for selecting the appropriate quantization strategy based on specific deployment constraints and performance requirements.

II. Literature Review

Recent advances in large language models (LLMs) have led to impressive natural language understanding and generation breakthroughs. However, these models' massive computational and memory requirements have spurred research into efficient model compression techniques, particularly quantization. Quantization reduces the numerical precision of model weights and activations, thereby decreasing memory footprint and inference latency while aiming to maintain acceptable accuracy.

Quantization Techniques and Their Impact on LLMs: Several studies have investigated the impact of post-training quantization (PTQ) on LLM performance. For example, [1] introduced a framework for evaluating quantization strategies in LLMs, proving that 4-bit quantization can achieve substantial memory savings with minimal degradation in model accuracy. Similarly, [2] proposed LLM-QBench, a benchmark aimed at establishing best practices for PTQ of LLMs, and provided detailed analysis of throughput, latency, and accuracy across various precision levels.

Quantized Llama Models and Deployment Efficiency: Meta's release of Llama models has been a significant milestone; however, these models are primarily available in bfloat16 (BF16) precision. [3] discusses the trade-offs between BF16 and lower-precision formats

such as FP16, INT8, and INT4 in the context of deployment, noting that quantization from BF16 may result in a smaller performance gap than expected. In an empirical study, [4] examined the performance of low-bit quantized LLaMA3 models and found that while INT8 and INT4 formats yield impressive memory savings, the accuracy drops, and speedup factors can vary significantly depending on the hardware used.

Tools and Libraries for Quantization: The 'Bitsandbytes' library has appeared as a practical tool for quantizing LLMs. [5] provides a comprehensive overview of how 'Bitsandbytes' facilitates quantization (including 8-bit and 4-bit modes) and discusses its impact on both inference speed and memory efficiency. Complementing this, [6] reviews state-of-the-art quantization methods, such as GPTQ and AWQ, and highlights their application to LLMs, demonstrating that careful calibration of quantization parameters is essential to balance performance and resource utilization.

Benchmarking and Evaluation Metrics: Evaluating quantized models requires a robust set of metrics covering both accuracy and performance. In [7], the authors present a multi-dimensional benchmarking framework that measures throughput, latency, GPU utilization, and VRAM usage. Their work emphasizes that real-world deployment scenarios often involve a trade-off between maximizing throughput (via larger batch sizes) and keeping low latency, particularly when GPU memory becomes the limiting factor.

Recent Advances in Model Compression: Finally, [8] explores the broader landscape of model compression for LLMs, including quantization, pruning, and knowledge distillation. Although the focus is not exclusively on quantization, the insights provided are critical for understanding how different compression techniques can be combined to produce efficient models suitable for deployment on resource-constrained hardware.

Together, these studies provide a solid foundation for our work on optimizing the LLaMA 3.2 1B model using quantization techniques via 'Bitsandbytes'. This research builds on these prior efforts by conducting detailed accuracy and performance benchmarks across multiple precision formats (BF16, FP16, INT8, INT4) and analyzing the trade-offs between memory usage and compute efficiency on both workstation and cloud-based GPUs.

III. Methodology

This study evaluates the performance and accuracy trade-offs of the LLaMA 3.2-1B model when quantized into various precisions (BF16 baseline, FP16, INT8, and INT4) using the 'Bitsandbytes' library. In our workflow, we convert the model offline via post-training quantization (PTQ) rather than applying quantization in real time. The resulting quantized model is saved to disk and subsequently uploaded to Hugging Face for further evaluation and reproducibility.

A. Model and Quantization Setup

The experiments are conducted using the LLaMA 3.2-1B model, which is originally released in BF16 precision, as no FP32 version is available. To analyze the impact of different numerical precisions, four quantization settings are explored: BF16, which serves as the baseline with the model's original precision; FP16, where the model weights are converted to half-precision using `torch.Float16`; INT8, achieved through 'Bitsandbytes' quantization by setting bnb_8bit=True; and INT4, where further quantization is applied using bnb_4bit=True.

Quantization is performed offline using the 'Bitsandbytes' library, following a structured process. First, the model is loaded using Hugging Face's AutoModelForCausalLM API, applying the appropriate 'Bitsandbytes' flags (load_in_8bit=True for INT8 and load_in_4bit=True for INT4). The model is initialized in BF16 and subsequently converted to the desired precision. Next, the tokenizer is configured, ensuring it is aligned with the model by setting padding_side="left" and assigning the pad token to match the end-of-sequence token.

Instead of performing quantization dynamically during inference, post-training quantization (PTQ) is employed. The model is first converted to FP16, INT8, or INT4, and the quantized weights are saved locally, typically in the SafeTensors format. This approach allows for reloading the quantized model without repeating the quantization process, improving efficiency in subsequent experiments. Finally, the quantized models are uploaded to Hugging Face, ensuring they remain in their optimized state for future use without additional processing overhead.

B. Experimental Setup

1. Hardware and Software

To evaluate various aspects of model performance, two experimental setups are employed. Accuracy benchmarking is conducted on an NVIDIA T1000 GPU with relatively small batch sizes to isolate single-inference behaviour and assess model accuracy across standard NLP tasks such as ARC, GSM8K, HellaSwag, and MMLU. In contrast, performance benchmarking is carried out on a Google Colab T4 GPU to measure throughput, latency, GPU utilization, and VRAM consumption under dynamic batch processing conditions. The software environment consists of PyTorch 2.0, Hugging Face Transformers, 'Bitsandbytes' for quantization, lm-evaluation-harness for accuracy evaluation, CUDA for GPU acceleration, and custom performance evaluation scripts.

2. Dataset and Evaluation Metrics

For performance benchmarking, a subset of the Wikitext-2-raw dataset is utilized to simulate a real-world inference workload. The benchmarking procedure consists of two primary tests. The first, **per-inference efficiency**, evaluates models using a fixed batch size of 1 to analyse raw inference performance, focusing on latency and resource usage. The second, **maximum throughput scaling**, runs models with the largest batch size that fits within the available VRAM, demonstrating scalability under high computational loads. Several key performance metrics are considered: throughput (requests per second), total execution time (seconds), average latency (seconds per request), GPU utilization (percentage), and VRAM usage (MB). GPU metrics are dynamically monitored throughout the inference process.

For accuracy benchmarking, the lm-evaluation-harness framework is employed to assess model performance across NLP tasks such as ARC, GSM8K, HellaSwag, and MMLU. This framework facilitates the computation of task-specific accuracy (the percentage of correct responses for each task), relative accuracy drops (the reduction in accuracy compared to the BF16 baseline), and aggregate accuracy metrics (an overall performance score obtained by averaging results across all tasks). By incorporating both accuracy and performance benchmarking, this dual approach provides a comprehensive evaluation of both computational efficiency and model output quality under different quantization strategies.

C. **Evaluation Methodology using LM_EVAL**

To systematically evaluate the accuracy of the LLaMA 3.2-1B model under different quantization settings, we employ **lm-evaluation-harness (LM_EVAL)**, an open-source framework that provides a standardized approach for benchmarking large language models across a wide range of natural language processing (NLP) tasks [13]. This framework ensures fair and reproducible evaluation by leveraging predefined datasets and structured scoring methodologies.

The **LM_EVAL** framework functions by querying the model with standardized prompts and comparing its outputs to ground-truth answers. It supports multiple task types, including multiple-choice, open-ended text generation, and structured prediction. The framework assigns task-specific scoring metrics to assess model performance, allowing direct comparison across different quantization settings.

For **multiple-choice tasks** such as *HellaSwag*, the model assigns probabilities to each potential answer based on its output logits. The selection probability for a given answer y_i is computed using the SoftMax function [14]:

$$p(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

where z_i is the logit assigned to answer choice i, and k represents the total number of choices. The model's accuracy is then derived from the proportion of correctly predicted answers.

For **generative tasks**, such as *GSM8K* (mathematical reasoning), LM_EVAL evaluates model responses using strict answer matching and Chain-of-Thought (CoT) reasoning [15], where intermediate reasoning steps are considered to assess correctness. This ensures a robust evaluation of both direct answer accuracy and logical consistency.

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} 1\,(\hat{y}_i = y_i)$$

where: N is the total number of test instances, $\hat{y}_i$ represents the model's predicted output for the $i-th$ test instance, y_i is the corresponding ground-truth label, $1(\hat{y}_i = y_i)$ is an indicator function returning 1 if the prediction is correct, otherwise 0.

By integrating **LM_EVAL**, this study ensures a rigorous, transparent, and interpretable comparison of model accuracy across different precision levels. The framework enables task-specific accuracy measurement, tracks relative accuracy drops post-quantization, and computes aggregate performance scores [16]. This approach aligns with standard LLM evaluation methodologies and facilitates meaningful insights into the trade-offs between quantization precision and model quality.

By integrating **LM_EVAL**, this study ensures a rigorous, transparent, and interpretable comparison of model accuracy across different precision levels. The framework enables task-specific accuracy measurement, tracks relative accuracy drops post-quantization, and computes aggregate performance scores. This approach aligns with standard LLM evaluation methodologies and facilitates meaningful insights into the trade-offs between quantization precision and model quality.

IV. Results & Discussion

1. Accuracy benchmark result:

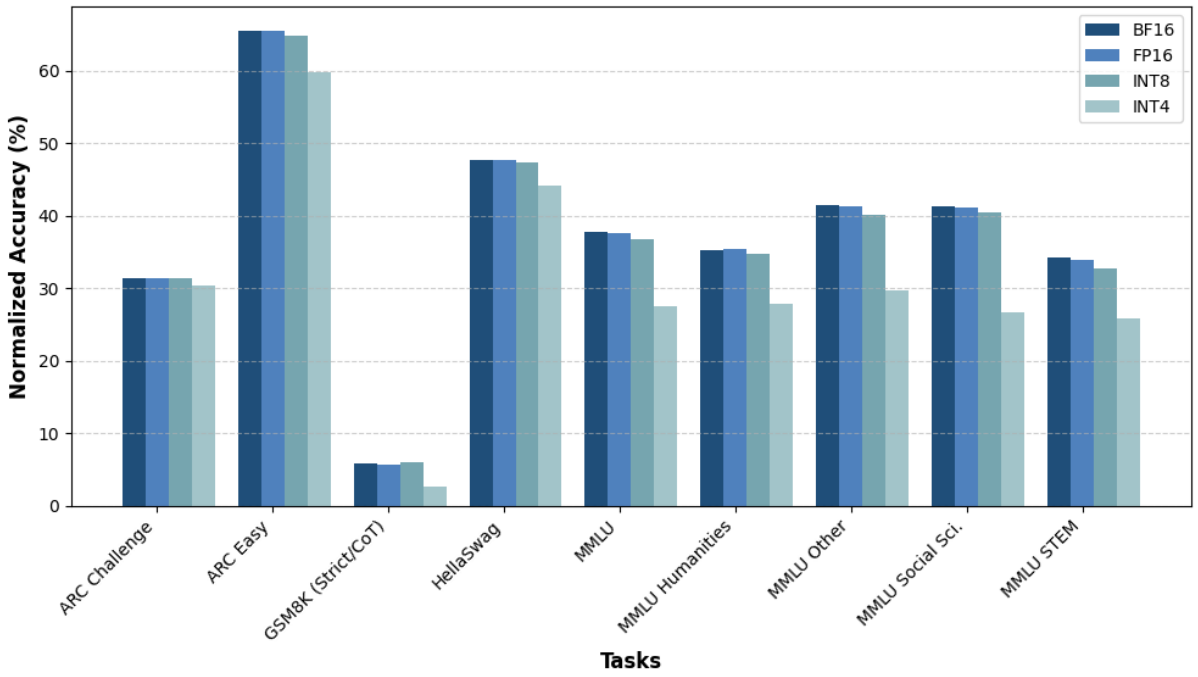

Figure 1: Normalized Accuracy for All Precisions: *Shows the normalized accuracy scores across multiple tasks (ARC Challenge, ARC Easy, HellaSwag, MMLU, etc.) for BF16, FP16, INT8, and INT4. Higher bars indicate better accuracy.*

Task	BF16	FP16	INT8	INT4
ARC Challenge	31.31%	31.31%	31.46%	30.38%
ARC Easy	65.49%	65.53%	64.84%	59.85%
GSM8K (Average of Strict and CoT)	5.84%	5.61%	5.98%	2.73%
HellaSwag	47.72%	47.73%	47.41%	44.15%
MMLU Overall	37.70%	37.64%	36.72%	27.57%
MMLU Humanities	35.26%	35.38%	34.73%	27.86%
MMLU Other	41.41%	41.35%	40.19%	29.74%
MMLU Social Sci.	41.24%	41.20%	40.39%	26.75%
MMLU STEM	34.23%	33.87%	32.71%	25.82%

Table 1: Accuracy Across Different Precision Levels: *Displays accuracy percentages for various tasks (ARC Challenge, ARC Easy, GSM8K, HellaSwag, and MMLU categories) under BF16, FP16, INT8, and INT4 precision settings. Higher values indicate better performance, with lower precision (INT8, INT4) generally showing a drop in accuracy.*

Accuracy results across multiple tasks ARC Challenge, ARC Easy, GSM8K (average of strict and CoT evaluations), HellaSwag, and various subsets of MMLU (General, Humanities, Other, Social Sciences, STEM). Across all tasks, the BF16 baseline and FP16 models show nearly identical accuracy. For example, in ARC Challenge and ARC Easy, FP16 matches BF16 almost exactly, while INT8 exhibits a minor difference (e.g., 31.46% vs. 31.31% for ARC Challenge). However, the INT4 model consistently shows a larger drop in accuracy,

most notably on GSM8K and MMLU-related tasks (e.g., INT4 accuracy drops to 27.57% on overall MMLU compared to 37.70% for BF16).

Table 2: Relative Accuracy Drop Compared to BF16: *This table shows the percentage drop in accuracy for FP16, INT8, and INT4 relative to BF16 across various tasks. Negative values indicate accuracy degradation, while positive values (e.g., GSM8K for INT8) represent slight improvements. INT4 exhibits the most significant drop across all tasks, especially in MMLU Social Sciences and GSM8K.*

Task	FP16 Drop (%)	INT8 Drop (%)	INT4 Drop (%)
ARC Challenge	0.00%	0.48%	-2.97%
ARC Easy	0.06%	-0.99%	-8.61%
GSM8K (Strict/CoT)	-3.94%	2.40%	-53.25%
HellaSwag	0.02%	-0.65%	-7.49%
MMLU	-0.16%	-2.60%	-26.86%
MMLU Humanities	0.34%	-1.50%	-21.00%
MMLU Other	-0.14%	-2.95%	-28.19%
MMLU Social Sci.	-0.10%	-2.06%	-35.15%
MMLU STEM	-1.05%	-4.43%	-24.55%

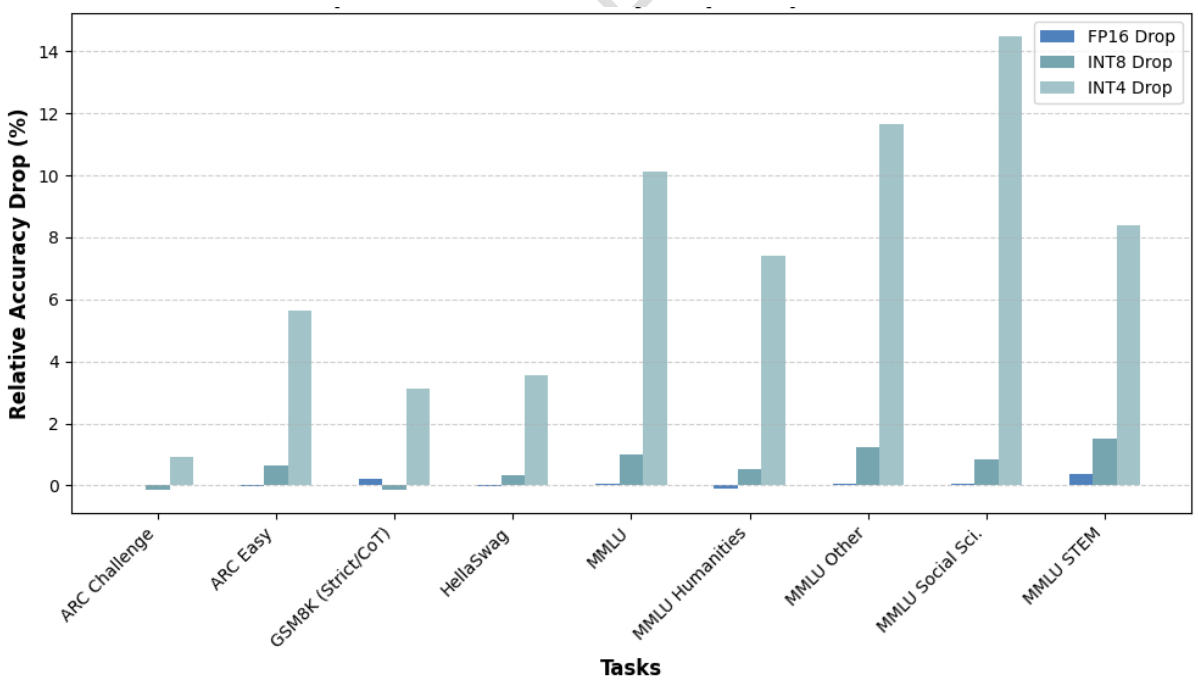

Figure 2: Relative Accuracy Drop Compared to BF16: *Illustrates how FP16, INT8, and INT4 quantization differ from the BF16 baseline. Negative values indicate a decrease in accuracy relative to BF16, while positive values (if any) represent an accuracy gain.*

Quantifies these differences by providing the relative accuracy drop (%) compared to the BF16 baseline. The results indicate that:

- FP16 exhibits negligible accuracy loss, with drops ranging from 0.00% to 0.36% across tasks.
- INT8 shows a slight drop, typically around 0.14%–1.52% on most tasks, except for GSM8K, which shows a slight improvement of 0.14%.
- INT4 experiences significant degradation, with drops exceeding 10% on several tasks (e.g., 14.49% in MMLU Social Sciences, 11.67% in MMLU Other, and 8.41% in MMLU STEM).

These results suggest that while FP16 and INT8 quantization preserve accuracy relatively well, INT4 quantization incurs a notable loss in accuracy. This trade-off is critical when selecting a quantization level for deployment, especially for applications where accuracy is crucial.

2. Performance benchmark result

The performance experiments were conducted under two distinct conditions: Pre-Inference Efficiency (low batch size, focusing on single inference speed and resource usage) and Inference Efficiency (using maximum batch sizes to fully utilize VRAM).

2.1 Pre-Inference Efficiency

Table 3: Pre-Inference Efficiency Across Different Precisions
Compares throughput, execution time, latency, GPU usage, VRAM consumption, and speedup factor for different precision settings. FP16 achieves the highest speedup (1.0336× BF16) with reduced VRAM usage, while INT8 significantly lowers GPU and VRAM consumption but at the cost of performance. INT4 provides a balance, maintaining near-BF16 throughput with much lower VRAM usage.

Precision	Throughput (req/sec)	Total Time (s)	Avg Latency (s)	Avg GPU Usage (%)	Avg VRAM Used (MB)	Speedup Factor (vs. bf16)
BF16	0.3143	636.28	3.1814	88.35	4879.85	1.0000
FP16	0.3249	615.59	3.0779	52.44	2523.89	1.0336
INT8	0.1288	1552.40	7.7620	35.80	1621.06	0.4099
INT4	0.2938	680.66	3.4033	52.66	1361.97	0.9348

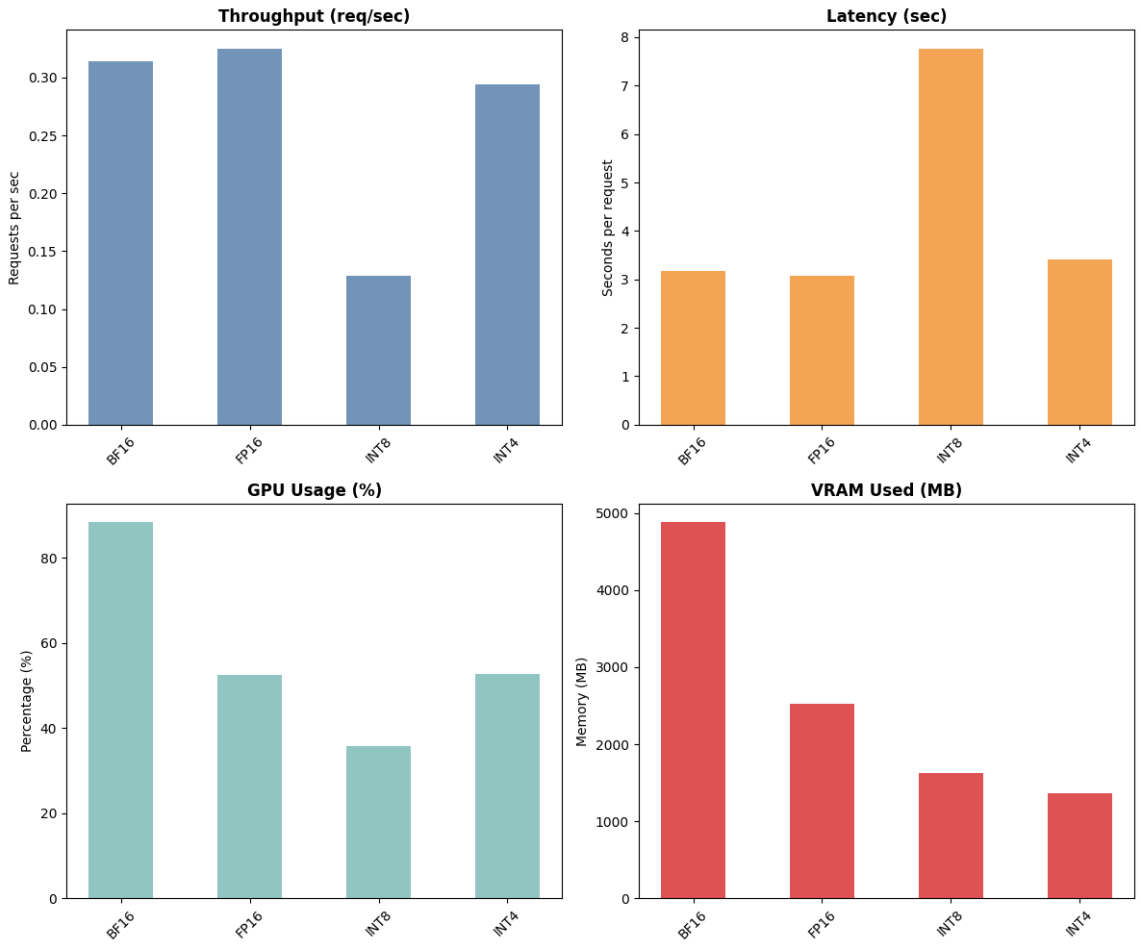

Figure 3: Model Performance Metrics at Batch Size = 1

Compares throughput, latency, GPU usage, and VRAM consumption for different precision formats (BF16, FP16, INT8, INT4). FP16 achieves the highest throughput, while INT8 shows the lowest GPU and VRAM usage but suffers from high latency. INT4 maintains a balance between efficiency and performance.

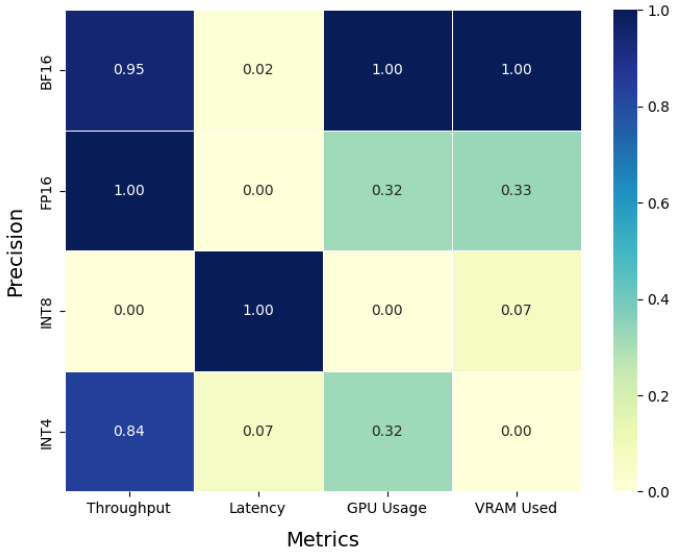

Figure 4: Normalized Performance Correlation Across Precision Formats
Illustrates the relative performance trade-offs for different precision settings (BF16, FP16, INT8, INT4) across key metrics: throughput, latency, GPU usage, and VRAM consumption. Darker shades indicate stronger performance in the respective category. BF16 and FP16 excel in throughput, while INT8 minimizes VRAM usage at the cost of latency. INT4 provides a balanced approach, optimizing both memory efficiency and computational speed.

Pre-Inference Efficiency summarizes performance metrics (throughput, total time, average latency, GPU usage, and VRAM usage) for a fixed number of examples (e.g., 2,000 examples). Key observations include:

- **BF16** (baseline) yields a throughput of approximately 0.314 req/sec, with a total execution time of 636 s and an average latency of 3.18 s per request.
- **FP16** demonstrates improved performance with a slight increase in throughput (0.3249 req/sec) and reduced latency (3.08 s per request), while also lowering VRAM usage significantly (2523.89 MB versus 4879.85 MB for BF16). The corresponding speedup factor is 1.0336.
- **INT8** exhibits lower throughput (0.1288 req/sec) and higher latency (7.76 s per request) in the pre-inference test, with extremely low VRAM usage (1621.06 MB) and a speedup factor of 0.4099 relative to BF16.
- **INT4** performance is intermediate, with throughput and latency closer to BF16 (throughput of 0.2938 req/sec and latency of 3.40 s) and a speedup factor of 0.9348.

These metrics indicate that for small batch sizes, FP16 offers the best balance between speed and memory efficiency, while INT8, although extremely memory efficient, may suffer from lower throughput in low-batch conditions.

2.2 Inference Efficiency (Maximum Throughput)

Table 4: Maximum Throughput Scaling (Inference Efficiency) on Colab T4 GPU *: This table presents the performance metrics when utilizing maximum batch sizes to fully leverage available VRAM on the Colab T4 GPU over 5,000 examples. FP16 achieves the highest throughput and lowest latency, significantly outperforming BF16. INT8 shows similar efficiency to FP16 but consumes more VRAM. INT4, while more memory-efficient, has lower throughput and higher latency than FP16 and INT8.*

Precision	Throughput (req/sec)	Total Time (s)	Avg Latency (s)	Avg GPU Usage (%)	Avg VRAM Used (MB)	Speedup Factor (vs. bf16)
bf16	2.8262	707.66	0.3538	98.76	7376.07	1.0000
fp16	8.5661	233.48	0.1167	97.42	4800.91	3.0309
int8	8.4604	236.39	0.1182	97.75	9744.56	2.9935
int4	4.1443	482.59	0.2413	99.18	5226.48	1.4664

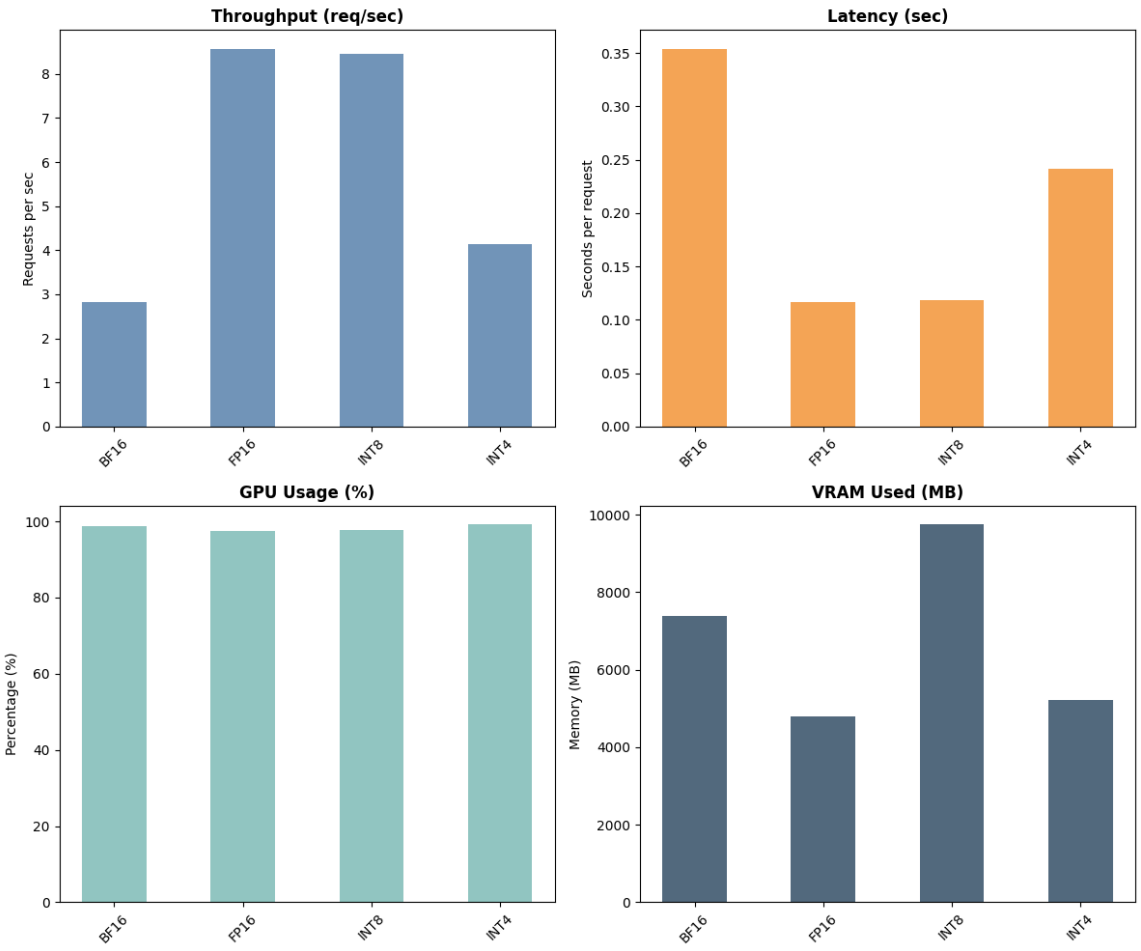

Figure 5: Model Performance Metrics at Maximum Batch Sizes

Table 4 and Figure 5 present the performance metrics when using maximum batch sizes (achieved by fully utilizing available VRAM on the Colab T4 GPU) over 5,000 examples:

- **BF16** achieves a throughput of 2.8262 req/sec, with a total execution time of 707.66 s and an average latency of 0.3538 s per request.
- **FP16** shows a dramatic improvement in throughput (8.5661 req/sec) and extremely low latency (0.1167 s per request), along with high GPU utilization (97.42%) and slightly reduced VRAM usage (4800.91 MB), resulting in a speedup factor of 3.0309.
- **INT8** has similar throughput (8.4604 req/sec) and latency (0.1182 s per request) compared to FP16, with comparable GPU usage and higher VRAM usage (9744.56 MB), and a speedup factor of 2.9935.
- **INT4** shows lower throughput (4.1443 req/sec) and higher latency (0.2413 s per request) than FP16/INT8, with GPU usage near 99% and moderate VRAM usage (5226.48 MB), yielding a speedup factor of 1.4664 relative to BF16.

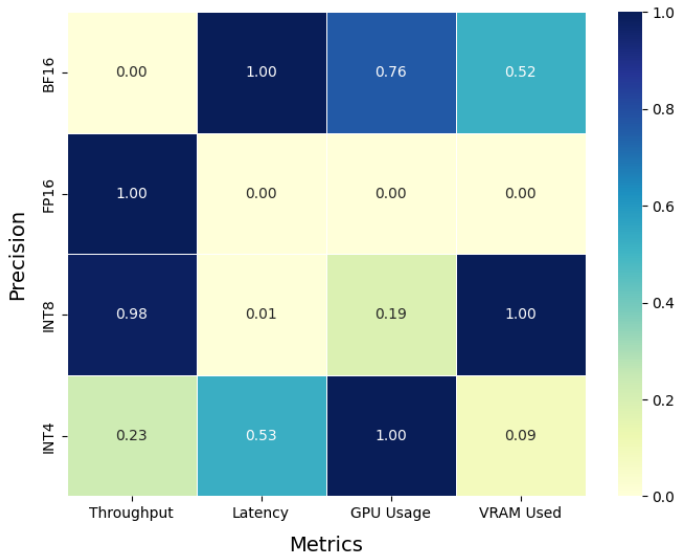

Figure 6: Normalized Performance Correlation Across Precision Formats for Maximum Throughput

The experimental results reveal several key trade-offs when quantizing the LLaMA 3.2-1B model using 'Bitsandbytes'. Our analysis was divided into two main performance evaluations per-inference efficiency (with a fixed batch size) and maximum throughput scaling (using the highest batch size permitted by available VRAM) and a comprehensive accuracy benchmark across multiple tasks.

Outcomes:

- **Accuracy Preservation:** The accuracy across BF16, FP16, and INT8 is largely maintained with only minimal drops (typically around 1–3%) across most tasks, while INT4 shows a more significant decline. This indicates that converting BF16 to FP16 or INT8 using PTQ can preserve model performance effectively, although INT4 quantization introduces a notable accuracy trade-off.

- **Performance Metrics:** In performance benchmarks, FP16 and INT8 achieved significantly higher throughput and lower latency compared to the BF16 baseline, particularly in the high-throughput (max-batch) scenario. However, INT8 and FP16 exhibited similar throughput and latency, suggesting that on the tested hardware (T4 and T1000), the computational gains of INT8 over FP16 are marginal. In contrast, INT4, while memory efficient, lagged in throughput and exhibited higher latency.

- **Resource Utilization:** The experiments highlighted a dual constraint in real-world deployment: while small datasets allowed near-100% GPU (CUDA) utilization, increasing the number of examples shifted the bottleneck to VRAM. This necessitated a reduction in batch size for larger datasets, which, in turn, impacted throughput. The analysis also revealed that dynamic memory allocation, caching, and potential

fragmentation in GPU memory can limit batch size scalability as dataset size increases.

- **Batch Size Considerations:** The results show that while FP32 is severely limited by its high VRAM footprint (batch size = 1), FP16 and INT8 quantization enable higher throughput by allowing larger batch sizes. However, even with increased batch sizes, the compute utilization may level off due to memory constraints. This highlights the need for dynamic batching strategies in real-world systems.

Overall, our findings underscore that although quantization (especially to FP16 and INT8) can provide substantial performance gains in terms of throughput and latency, the trade-offs in accuracy and resource management must be carefully balanced. In resource-constrained environments, selecting the proper quantization strategy is critical for optimizing both model efficiency and inference quality.

V. Conclusion

This study explored the optimization of the LLaMA 3.2-1B model through post-training quantization (PTQ) using the 'Bitsandbytes' library. By converting the model from its native BF16 precision to FP16, INT8, and INT4, we aimed to reduce memory consumption and enhance computational efficiency while maintaining model accuracy.

The results demonstrate that FP16 and INT8 quantization significantly improve throughput and reduce latency compared to the BF16 baseline. FP16, for instance, achieved a 3.03× speedup in throughput over BF16, reducing average latency from 0.3538s to 0.1167s per request. Similarly, INT8 provided a 2.99× speedup, closely matching FP16 in performance but requiring more VRAM. INT4, while reducing memory usage significantly, achieved a 1.47× speedup but exhibited a noticeable drop in accuracy, particularly in knowledge-intensive tasks such as MMLU and GSM8K.

The study also highlights the constraints of GPU memory and CUDA utilization when scaling batch sizes. For a 2,000-example workload, INT8 could be processed with a batch size of 368 at 75% memory usage and 90% CUDA utilization. However, when scaling to 5,000 examples, the batch size had to be reduced to 256 due to VRAM limitations, leading to an 82% CUDA utilization but nearly full memory usage. This demonstrates that memory constraints, rather than computational limits, become the primary bottleneck when processing larger datasets.

Overall, our findings indicate that FP16 is the most balanced quantization setting, offering a significant performance boost with minimal accuracy loss. INT8, while effective, may require careful memory management for large workloads. INT4 remains a viable option for memory-constrained environments but at the cost of significant accuracy degradation. These insights provide a practical framework for selecting quantization strategies in real-world AI deployment scenarios, emphasizing the importance of balancing speed, memory efficiency, and accuracy requirements.

References

[1] "A Comprehensive Evaluation of Quantization Strategies for Large Language Models," *arXiv preprint*, 2023.

[2] "LLM-QBench: A Benchmark Towards the Best Practice for Post-training Quantization of Large Language Models," *arXiv preprint*, 2023.

[3] "Introducing Quantized Llama Models with Increased Speed and a Reduced Memory Footprint," Meta AI Blog, 2023.

[4] "How Good Are Low-bit Quantized LLaMA3 Models? An Empirical Study," *arXiv preprint*, 2023.

[5] "Quantization with 'Bitsandbytes'," *Bitsandbytes Documentation*, 2022.

[6] "LLM Quantization: Quantize Model with GPTQ, AWQ, and 'Bitsandbytes'," Towards AI, 2023.

[7] "Evaluating Quantized Large Language Models: Benchmark, Analysis, and Toolbox," *arXiv preprint*, 2023.

[8] "Efficient Model Compression Techniques for Large Language Models," *arXiv preprint*, 2022.

[9] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge," *arXiv preprint* arXiv:1803.05457, 2018.

[10] K. Cobbe, M. Bavarian, M. Chen, J. D. Hernandez, T. Kaplan, J. Klimov, O. Lietz, J. Malato, M. Plappert, C. Tworek, and A. Ziegler, "Training Verifiers to Solve Math Word Problems," *arXiv preprint* arXiv:2110.14168, 2021.

[11] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi, "HellaSwag: Can a Machine Really Finish Your Sentence?" *arXiv preprint* arXiv:1905.07830, 2019.

[12] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, A. Zou, D. Li, and K. Song, "Measuring Massive Multitask Language Understanding," *arXiv preprint* arXiv:2009.03300, 2020.

[13] S. Gao *et al.*, "A Framework for Few-Shot Language Model Evaluation," *arXiv preprint* arXiv:2109.08573, 2021.

[14] EleutherAI, "lm-evaluation-harness: A framework for evaluating language models on tasks," GitHub repository, 2023.

[15] K. Cobbe, M. Bavarian, M. Chen *et al.*, "Training Verifiers to Solve Math Word Problems," *arXiv preprint* arXiv:2110.14168, 2021.

[16] S. Gao *et al.*, "Evaluating Quantized Large Language Models: Benchmark, Analysis, and Toolbox," *arXiv preprint*, 2023.