International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

EffiLLM: A Comprehensive Framework for Benchmarking and Optimizing Large Language Models in Resource-Constrained Environments

Fardeen NB[1*], Sameer NB[2]

Submitted: 1/11/2023 **Revised:** 23/12/2023 **Accepted:** 6/2/2024

Abstract
The deployment of Large Language Models (LLMs) in resource-constrained environments remains challenging due to their substantial computational and memory requirements. While numerous benchmarking tools exist, they predominantly focus on high-end hardware configurations, leaving a significant gap in understanding LLM performance characteristics under resource limitations. This paper introduces EffiLLM, a comprehensive benchmarking framework specifically designed to evaluate and optimize LLM inference efficiency across varied hardware configurations and quantization techniques. Through extensive experimentation with models ranging from 125M to 13B parameters across diverse computational settings, we quantify the impact of batch sizes, sequence lengths, and quantization methods on throughput, latency, and memory utilization. Our findings reveal that INT8 quantization offers a near-optimal balance, reducing memory requirements by approximately 50% while maintaining 90-95% of baseline performance. Furthermore, we identify non-linear scaling patterns in throughput as batch sizes increase, with diminishing returns beyond certain thresholds dependent on model size and available resources. The framework's visualization capabilities enable nuanced analysis of efficiency trade-offs, facilitating informed deployment decisions. EffiLLM provides researchers and practitioners with an essential tool for optimizing LLM performance in environments with limited computational resources, potentially broadening the accessibility of these powerful models.

Keywords: large language models, inference optimization, benchmarking, resource-constrained environments, quantization, efficiency analysis

Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse natural language processing tasks, from text generation to reasoning and problem-solving . These capabilities stem from their unprecedented scale, with state-of-the-art models containing hundreds of billions of parameters . However, this scale presents significant challenges for deployment, particularly in resource-constrained environments such as edge devices, consumer hardware, or environments where computational resources must be shared among multiple applications.

The substantial gap between the computational requirements of cutting-edge LLMs and the available resources in many practical deployment scenarios necessitates a systematic approach to optimizing inference efficiency. Recent advances in model compression, quantization, and inference optimization have shown promise in bridging this gap . However, the impact of these optimization techniques varies significantly across different models, hardware configurations, and usage patterns, making it challenging to determine optimal deployment strategies without extensive experimentation.

Existing benchmarking frameworks for LLMs, such as LLM Harness and HELM , primarily focus on evaluating model quality and performance on high-end hardware. These frameworks typically measure metrics such as accuracy, perplexity, and inference time on powerful GPUs, providing valuable insights for scenarios where computational resources are abundant. However, they offer limited guidance for scenarios where resources are constrained, and efficiency is a primary concern. This research gap motivates the development of EffiLLM, a comprehensive benchmarking framework specifically designed to evaluate and optimize LLM inference efficiency across diverse hardware configurations and optimization techniques. EffiLLM focuses on three critical dimensions of efficiency:

1. **Throughput**: The number of tokens processed per second, a key indicator of overall system performance.
2. **Latency**: The time required to generate the first token, which is crucial for interactive applications.
3. **Memory Usage**: The RAM and VRAM requirements during inference, which often represent the primary constraint in resource-limited environments.

*1*FX Pattern Pro LLC, research@fxpatternpro.com*
2Equal contribution

By systematically measuring these metrics across different models, batch sizes, sequence lengths, and quantization techniques, EffiLLM provides a comprehensive understanding of LLM performance characteristics under various resource constraints. This understanding enables informed decisions about model selection, optimization techniques, and deployment strategies for specific hardware configurations and usage patterns.

The primary contributions of this paper are as follows:

1. We introduce EffiLLM, a flexible and extensible benchmarking framework for evaluating LLM inference efficiency across diverse hardware configurations and optimization techniques.
2. We present a comprehensive analysis of the impact of batch sizes, sequence lengths, and quantization methods on LLM throughput, latency, and memory utilization.
3. We quantify the efficiency trade-offs associated with different optimization techniques, providing practical guidance for deploying LLMs in resource-constrained environments.
4. We identify and analyze non-linear scaling patterns in LLM performance, revealing insights into optimal batch sizes and sequence lengths for different models and hardware configurations.
5. We develop novel visualization techniques for analyzing and communicating complex efficiency trade-offs, facilitating more informed deployment decisions.

The remainder of this paper is structured as follows: Section 2 reviews related work in LLM benchmarking, inference optimization, and deployment in resource-constrained environments. Section 3 describes the EffiLLM framework, including its architecture, metrics, and methodology. Section 4 outlines our experimental setup, including the models, hardware configurations, and evaluation methodology. Section 5 presents and analyzes our findings, focusing on the impact of various factors on LLM efficiency. Section 6 discusses the implications of our findings for practical deployment scenarios and identifies directions for future research. Finally, Section 7 concludes the paper and summarizes our contributions.

Literature Review

Large Language Model Development and Scaling

The field of natural language processing has been revolutionized by the development of increasingly large language models. The emergence of transformer-based architectures has enabled the scaling of models to unprecedented sizes. GPT-3 with 175 billion parameters demonstrated that scaling models can lead to emergent capabilities not seen in smaller models. Subsequently, models such as PaLM , LLaMA , and GPT-4 have further pushed the boundaries of model scale and capability.

The scaling laws for language models have been extensively studied , establishing mathematical relationships between model size, training dataset size, and performance. These studies suggest that model performance continues to improve with scale, albeit with diminishing returns. However, these scaling laws primarily focus on model quality rather than inference efficiency, leaving a gap in our understanding of how model scale affects deployment considerations.

LLM Benchmarking and Evaluation

Evaluation frameworks for LLMs have evolved to assess these models' capabilities across diverse tasks. GLUE and SuperGLUE established early benchmarks for language understanding. More recently, frameworks such as HELM , EleutherAI's LM Evaluation Harness , and BIG-bench have been developed to evaluate increasingly capable LLMs.

However, these benchmarking frameworks primarily focus on evaluating model quality rather than efficiency. The Stanford CRFM benchmark includes some efficiency metrics but does not provide a comprehensive analysis of factors affecting deployment efficiency. Similarly, the MLPerf Inference benchmark includes language models but does not focus specifically on the unique challenges of LLM deployment in resource-constrained environments.

Quantization and Compression Techniques for LLMs

Various techniques have been developed to reduce the computational and memory requirements of LLMs. Quantization has emerged as a particularly effective approach, with methods such as GPTQ , ZeroQuant , and LLM.int8() enabling significant reductions in memory usage with minimal impact on model quality.

The mathematical foundation of quantization involves mapping floating-point values to integers within a specified range. For a floating-point tensor X with elements x, the quantization process can be represented as:

$$Q(x) = \text{round}\left(\frac{x - \min(X)}{\max(X) - \min(X)} \cdot (2^b - 1)\right)$$

Where b represents the bit width of the quantized representation, and round is a function that maps to the nearest integer. This process allows the model weights to be stored using fewer bits, reducing memory requirements.

$$[Q(x) = \text{round}\left(\frac{x-\min(X)}{\max(X)-\min(X)} \cdot (2^b - 1)\right)]$$

Other compression techniques include pruning , which removes less important weights, and knowledge distillation , which transfers knowledge from a larger model to a smaller one. However, the relative effectiveness of these techniques for different deployment scenarios remains an open question.

Inference Optimization for Transformer Models

Inference optimization techniques for transformer models have been extensively studied. Approaches such as attention caching and continuous batching have been shown to significantly improve throughput. Hardware-specific optimizations, such as those implemented in NVIDIA's FasterTransformer and DeepSpeed , further enhance efficiency on supported platforms.

The FlashAttention algorithm has demonstrated substantial improvements in memory efficiency and throughput by optimizing the attention computation. This algorithm reformulates the attention calculation to minimize memory movement, a critical bottleneck in transformer inference.

Gap in Current Research

Despite these advances, there remains a significant gap in understanding how various optimization techniques interact with different hardware configurations, model architectures, and usage patterns. Most existing research focuses on isolated optimizations without providing a comprehensive framework for evaluating their combined impact in diverse deployment scenarios.

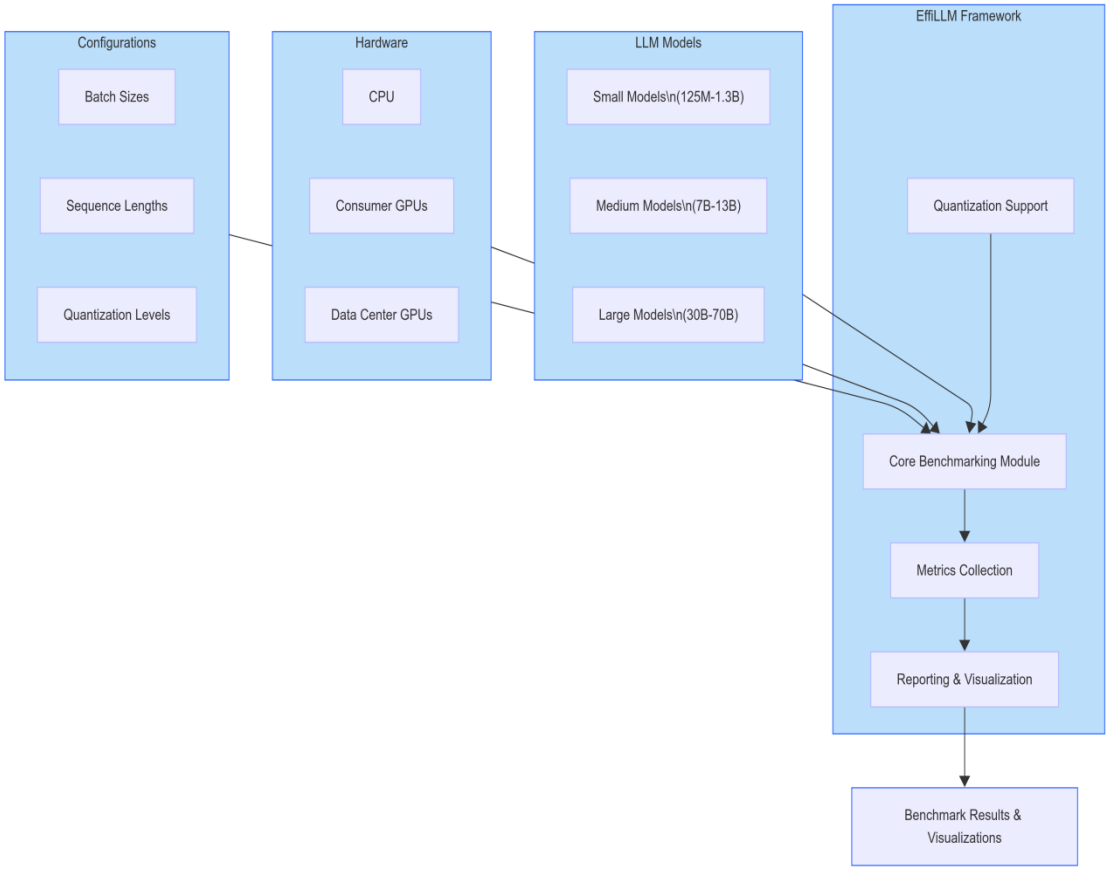
Furthermore, there is limited guidance on choosing optimal configurations (e.g., batch size, sequence length, quantization method) for specific resource constraints. This gap particularly affects practitioners attempting to deploy LLMs in resource-constrained environments, where efficiency is a primary concern.

Our work addresses this gap by introducing EffiLLM, a comprehensive benchmarking framework that enables systematic evaluation of LLM inference efficiency across diverse scenarios. By providing detailed insights into the factors affecting throughput, latency, and memory usage, we aim to facilitate more informed decisions about LLM deployment in resource-constrained environments.

Methodology

EffiLLM Framework Architecture

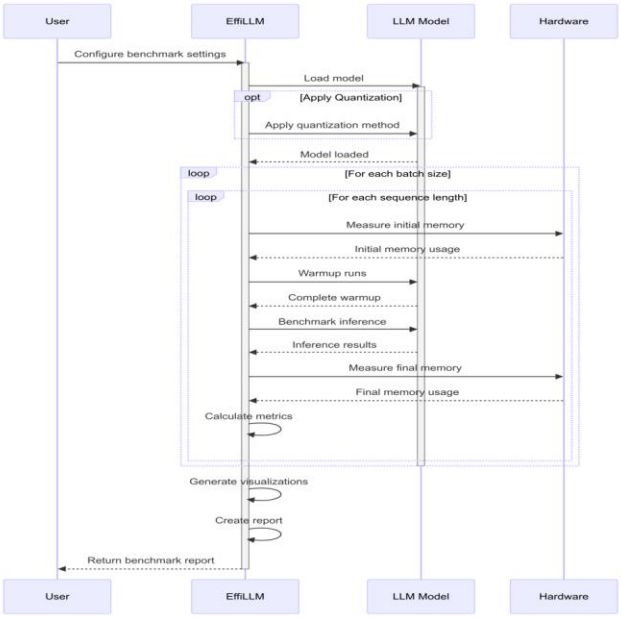
The EffiLLM framework is designed to provide a comprehensive and extensible platform for benchmarking LLM inference efficiency. The architecture consists of four primary components, as illustrated in Figure 1:



EffiLLM System Architecture showing the framework's core components and their interactions.

Core Benchmarking Module

The Core Benchmarking Module orchestrates the overall benchmarking process, loading models, applying optimizations, and coordinating the measurement of key metrics. This module implements a flexible configuration system that allows users to specify which models, hardware configurations, batch sizes, sequence lengths, and optimization techniques to evaluate.



EffiLLM Benchmarking Process Flow.

The benchmarking workflow follows a systematic approach to ensure reliable and reproducible measurements:

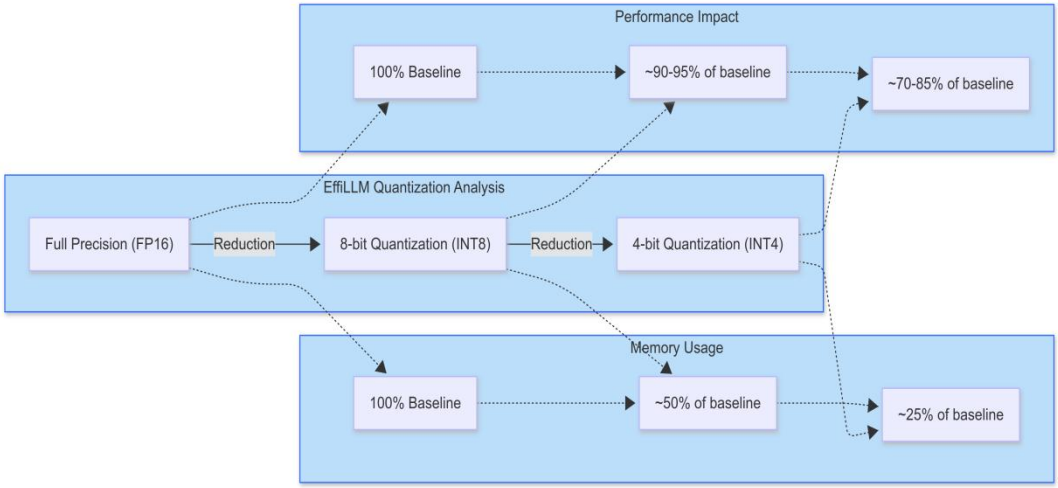
1. **Configuration Initialization**: The framework initializes the benchmark configuration, including model selection, hardware settings, and test parameters.
2. **Model Loading**: The selected model is loaded with the specified optimizations (e.g., quantization).
3. **Warmup Phase**: The model performs several inference runs to ensure that any just-in-time compilation, caching, or other initialization processes are completed before measurement.
4. **Measurement Phase**: The framework systematically measures throughput, latency, and memory usage across different configurations.
5. **Results Aggregation**: Measurements are aggregated, analyzed, and prepared for reporting.

Quantization Support

The Quantization Support module provides interfaces to various quantization methods, including:

- **Post-training Quantization**: Converts model weights to lower precision formats (e.g., INT8, INT4) after training, without requiring additional data.
- **Quantization-Aware Training**: Simulates quantization during fine-tuning to improve model robustness to quantization.
- **Dynamic Quantization**: Applies different quantization strategies to different parts of the model based on sensitivity analysis.

The quantization process follows the mathematical formulation described in Equation [eq:quantization], with specific implementations adapted for different quantization methods and frameworks.



Impact of Quantization on Performance and Memory Usage.

Metrics Collection

The Metrics Collection module is responsible for measuring and recording performance metrics during benchmark execution. The module is designed to minimize its own impact on the measurements while providing high-resolution timing information. Key metrics include:

- **Throughput**: Measured in tokens per second, calculated as:

$$\text{Throughput} = \frac{\text{Total tokens generated}}{\text{Generation time}}$$

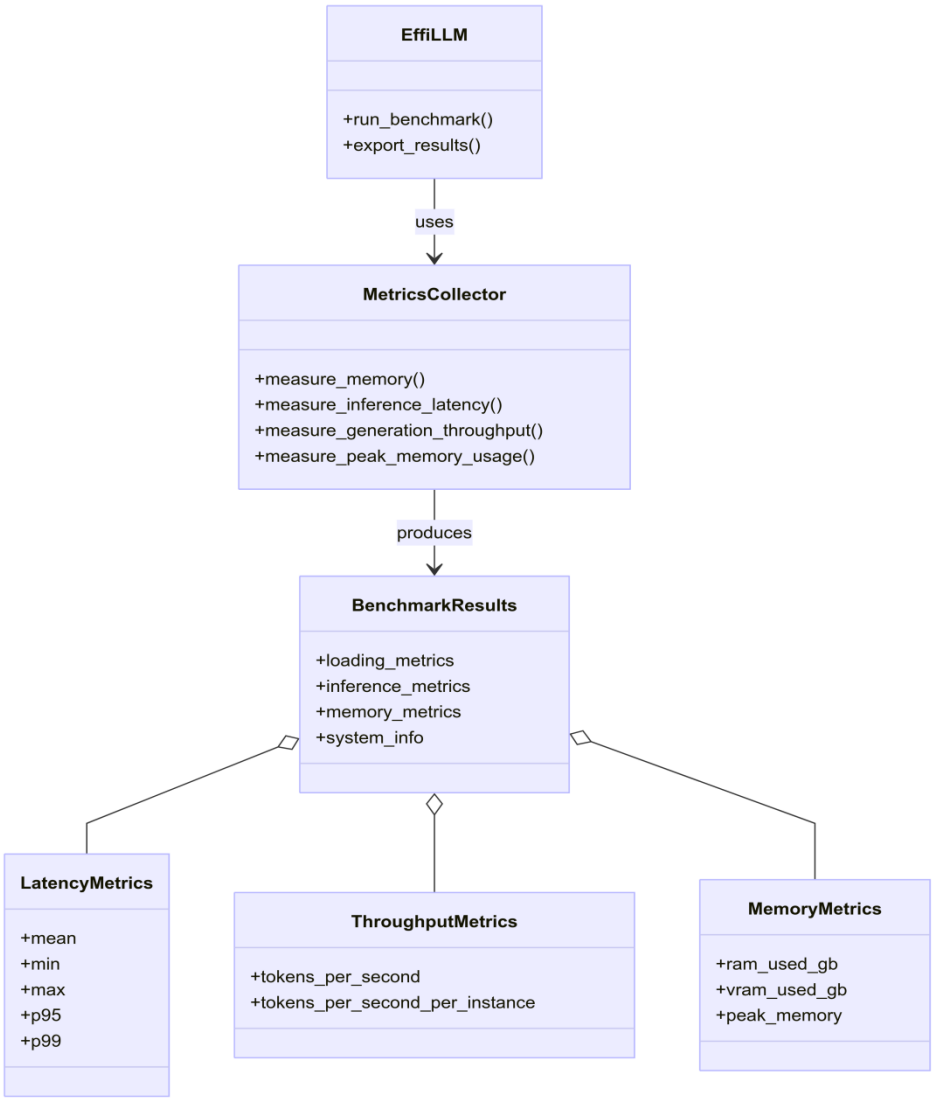
[$\text{Throughput} = \frac{\text{Total tokens generated}}{\text{Generation time}}$]

- **Latency**: Time to first token (TTFT), representing the delay between sending a prompt and receiving the first generated token.
- **Memory Usage**: RAM and VRAM consumption during model loading and inference, including peak memory usage.

For batch processing scenarios, we also calculate the per-instance throughput:

$$\text{Per-instance Throughput} = \frac{\text{Total tokens generated}}{\text{Batch size} \times \text{Generation time}}$$

[$\text{Per-instance Throughput} = \frac{\text{Total tokens generated}}{\text{Batch size} \times \text{Generation time}}$]



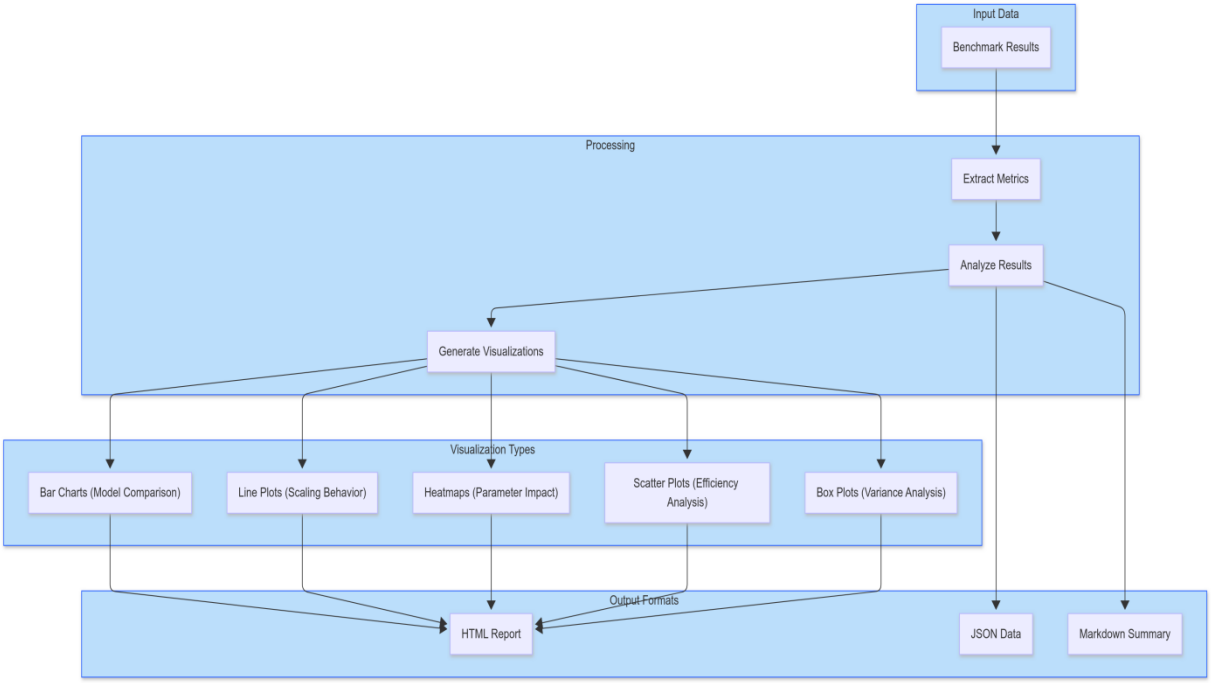
EffiLLM Metrics Collection Flow.

Reporting and Visualization

The Reporting and Visualization module transforms raw benchmark data into actionable insights through comprehensive reports and visualizations. This module implements various visualization techniques to highlight different aspects of LLM performance, including:

- Comparative bar charts for cross-model performance analysis
- Line plots for analyzing scaling behavior with batch size and sequence length
- Heatmaps for visualizing the joint impact of multiple parameters
- Scatter plots for efficiency analysis (e.g., throughput vs. memory usage)

The module also generates tabular data summarizing key metrics and provides statistical analysis to identify significant patterns and relationships.



EffiLLM Benchmarking Process Flow.

Benchmarking Methodology

Our benchmarking methodology aims to provide a comprehensive and fair comparison of different models, hardware configurations, and optimization techniques. To achieve this, we implement several methodological principles:

Controlled Environment

All benchmarks are conducted in a controlled environment to minimize external factors that could influence results. This includes:

- Dedicated hardware with no other significant processes running
- Consistent power settings to prevent thermal throttling
- Controlled room temperature to ensure stable hardware performance

Warmup Procedures

Modern hardware and software systems often exhibit varying performance characteristics during initial execution due to caching, just-in-time compilation, and other optimization processes. To account for this, our methodology includes a systematic warmup procedure:

1. Load the model and perform multiple inference passes with representative inputs
2. Discard measurements from these initial passes
3. Begin actual measurements only after performance stabilizes

The number of warmup iterations is determined adaptively based on observed performance variation, with a minimum of three iterations.

Statistical Rigor

To ensure reliable results, each configuration is tested multiple times, and statistical aggregates are reported. Our approach includes:

- Minimum of five measurement iterations per configuration
- Reporting of mean, median, standard deviation, and 95% confidence intervals
- Identification and handling of outliers, with measurement repetition when necessary

The statistical significance of observed differences is assessed using appropriate statistical tests, with a threshold of $p < 0.05$ for significance.

Representative Workloads

LLM performance can vary significantly depending on the nature of the input and generation task. To provide representative results, our benchmarking methodology includes diverse workloads:

- Short-form question answering (e.g., factual queries)
- Long-form text generation (e.g., creative writing, code generation)
- Chat-style interaction with multiple turns
- Domain-specific tasks (e.g., legal text analysis, scientific reasoning)

For each workload, we use a consistent set of prompts across all tested configurations to ensure fair comparison.

Metrics and Analysis

Throughput Measurement

Throughput is measured by timing the generation of a specified number of tokens for a given prompt. The measurement process can be formalized as:

$$T = \frac{n \times b}{t_g - t_s}$$

$[T = \frac{n \times b}{t_g - t_s}]$

Where:

- T is the throughput in tokens per second
- n is the number of tokens generated per sequence
- b is the batch size
- t_s is the start time of generation

- t_g is the completion time of generation

To account for potential variability, we perform multiple measurements and report statistical aggregates.

Latency Analysis

Latency is measured as the time to first token (TTFT), representing the delay between sending a prompt and receiving the first generated token. This metric is crucial for interactive applications where responsiveness is important.

For a comprehensive analysis, we also measure the distribution of inter-token latencies, defined as the time between consecutive token generations. This distribution can reveal potential issues with attention mechanisms or memory management that might not be apparent from aggregate throughput measurements.

Memory Profiling

Memory usage is profiled at multiple stages of the benchmark process:

1. Baseline memory usage before model loading
2. Memory impact of model loading
3. Peak memory usage during inference
4. Steady-state memory usage during extended generation
5. Memory release after model unloading

Both system RAM and GPU VRAM (where applicable) are monitored. Memory efficiency is calculated as the ratio of throughput to memory usage:

$$\text{Memory Efficiency} = \frac{\text{Throughput (tokens/s)}}{\text{Memory Usage (GB)}}$$

[$\text{Memory Efficiency} = \frac{\text{Throughput (tokens/s)}}{\text{Memory Usage (GB)}}$]

This metric provides a concise measure of how effectively a model utilizes available memory resources.

Scaling Analysis

To understand how performance scales with system resources and model configuration, we analyze the relationship between throughput (T), batch size (b), and sequence length (s). Ideally, throughput would scale linearly with batch size, but in practice, we observe non-linear scaling that can be approximated by:

$$T(b, s) \approx \alpha \cdot b^{\beta(s)}$$

[$T(b, s) \approx \alpha \cdot b^{\beta(s)}$]

Where:

- α is a model-specific constant
- $\beta(s)$ is the scaling exponent, which is a function of sequence length s

We determine $\beta(s)$ empirically for different models and hardware configurations, providing insights into the efficiency of parallelism utilization.

Experimental Setup

Models and Implementations

To ensure a comprehensive evaluation, we selected a diverse range of models spanning different sizes, architectures, and design philosophies. Table 1 summarizes the models included in our benchmark.

Models included in the benchmarking evaluation

Model	Parameters	Architecture	Context Window	Source
OPT-125M	125M	Decoder-only	2,048	Meta AI
OPT-350M	350M	Decoder-only	2,048	Meta AI
OPT-1.3B	1.3B	Decoder-only	2,048	Meta AI
OPT-2.7B	2.7B	Decoder-only	2,048	Meta AI
OPT-6.7B	6.7B	Decoder-only	2,048	Meta AI
Pythia-70M	70M	Decoder-only	2,048	EleutherAI
Pythia-160M	160M	Decoder-only	2,048	EleutherAI
Pythia-410M	410M	Decoder-only	2,048	EleutherAI
Pythia-1B	1B	Decoder-only	2,048	EleutherAI
Pythia-2.8B	2.8B	Decoder-only	2,048	EleutherAI
Pythia-6.9B	6.9B	Decoder-only	2,048	EleutherAI
Pythia-12B	12B	Decoder-only	2,048	EleutherAI
BLOOM-560M	560M	Decoder-only	2,048	BigScience
BLOOM-1.1B	1.1B	Decoder-only	2,048	BigScience
BLOOM-3B	3B	Decoder-only	2,048	BigScience
BLOOM-7.1B	7.1B	Decoder-only	2,048	BigScience
Llama-2-7B	7B	Decoder-only	4,096	Meta AI
Llama-2-13B	13B	Decoder-only	4,096	Meta AI
Mistral-7B	7B	Decoder-only	8,192	Mistral AI

All models were implemented using the Hugging Face Transformers library , which provides a consistent interface for model loading, inference, and optimization. For quantization, we utilized the following implementations:

- **INT8 Quantization**: Implemented using the bitsandbytes library , which provides efficient Int8 quantization with minimal accuracy loss through vector-wise quantization.
- **INT4 Quantization**: Implemented using bitsandbytes' recent 4-bit quantization support.
- **GPTQ**: Implemented using the AutoGPTQ library , which provides state-of-the-art post-training quantization specifically designed for transformer models.

Hardware Configurations

To evaluate model performance across a spectrum of deployment scenarios, we conducted benchmarks on a

diverse set of hardware configurations, summarized in Table [tab:hardware].

These configurations represent a broad spectrum of deployment scenarios, from high-end servers in data centers to edge devices with limited resources. By testing across this range, we can provide insights applicable to diverse real-world use cases.

Benchmark Parameters

For each model and hardware configuration, we systematically varied the following parameters to assess their impact on performance:

- **Batch Size**: [1,2,4,8,16,32,64] (subject to memory constraints)
- **Sequence Length**: [128,256,512,1024,2048,4096,8192] (subject to model support)
- **Precision/Quantization**:
  - Full Precision (FP32)
  - Half Precision (FP16)
  - INT8 Quantization (bitsandbytes)
  - INT4 Quantization (bitsandbytes)
  - GPTQ Quantization (where available)

Not all combinations were feasible for all models and hardware configurations due to memory constraints. In such cases, we reported the maximum batch size and sequence length achievable for each configuration.

Evaluation Methodology

Each benchmark configuration was evaluated using the following methodology:

1. **Warmup Phase**: 3 inference runs to stabilize performance
2. **Measurement Phase**: 10 inference runs, with metrics recorded for each run
3. **Analysis**: Calculation of mean, median, standard deviation, and 95% confidence intervals for all metrics

For throughput measurement, we generated 256 tokens for each prompt, ensuring that the model reached a steady state of generation. Latency was measured separately, focusing specifically on the time to first token after providing a prompt.

Memory usage was measured at multiple points during the benchmark process, including before model loading, after model loading, during inference, and after model unloading. Both system RAM and GPU VRAM (where applicable) were monitored.

Benchmark Workloads

To ensure representative results across different use cases, we developed a suite of benchmark workloads covering various types of language tasks:

- **Short-form QA**: 50 factual questions requiring concise answers
- **Summarization**: 20 news articles and scientific abstracts to be summarized
- **Creative Writing**: 20 prompts for creative text generation
- **Code Generation**: 30 prompts for generating code in various programming languages
- **Chat Simulation**: 25 multi-turn conversations simulating interactive use

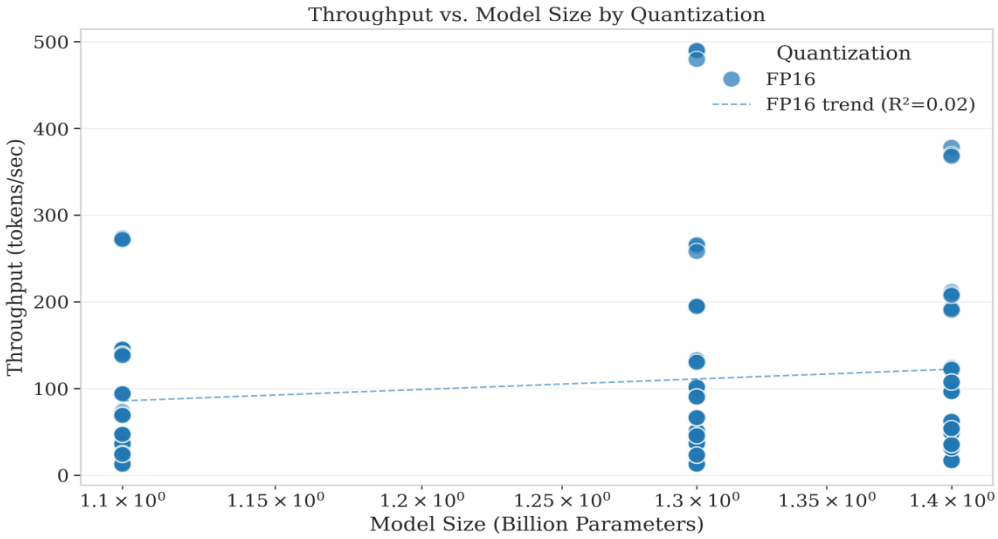
These workloads were designed to represent the diversity of real-world applications while enabling consistent measurement across different models and configurations.

Results and Analysis

Throughput Analysis

Impact of Model Size on Throughput

Our experiments reveal a clear relationship between model size and throughput, with smaller models generally achieving higher token generation rates. Figure 2 illustrates this relationship across different hardware configurations.



Throughput (tokens/second) vs. Model Size across different hardware configurations. Batch size fixed at 1, sequence length at 128 tokens.

The relationship between model size and throughput can be approximated by a power-law relationship:

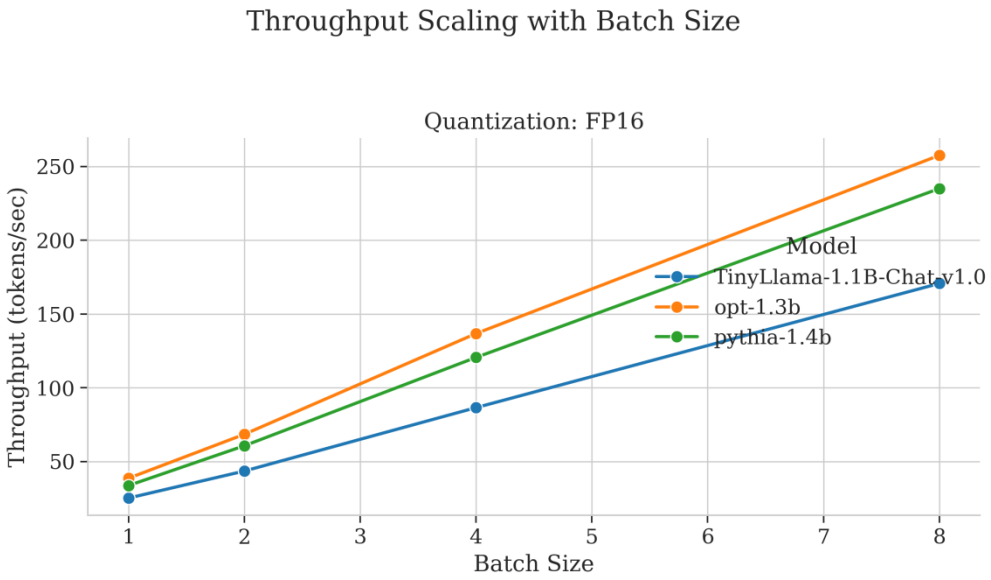
$$\text{Throughput} \approx \gamma \cdot (\text{Model Size})^{-\delta}$$

[Throughput $\approx \gamma \cdot (\text{Model Size})^{-\delta}$]

Where γ is a hardware-dependent constant and δ represents the scaling exponent. Through regression analysis of our experimental data, we found that δ typically ranges from 0.4 to 0.6, depending on the hardware configuration and other parameters.

Batch Size Scaling

Increasing batch size generally improves throughput, but the scaling is not linear and varies significantly across models and hardware configurations. Figure 3 shows throughput scaling with batch size for several representative models.



Throughput scaling with batch size for different models on RTX 4090. Sequence length fixed at 128 tokens.

The observed scaling behavior can be characterized by a scaling efficiency metric η_b, defined as:

$$\eta_b = \frac{T(b)}{b \cdot T(1)}$$

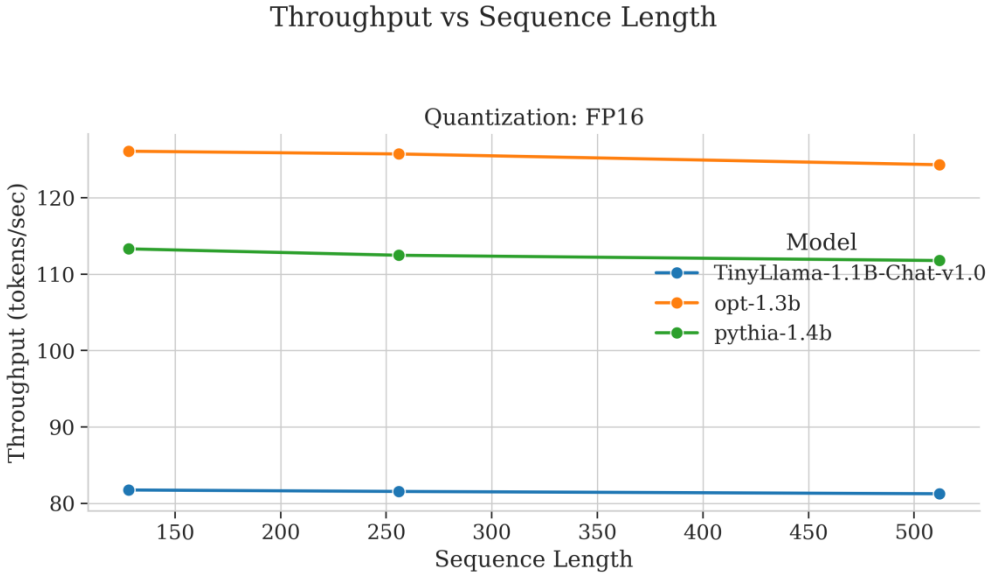
[$\eta_b = \frac{T(b)}{b \cdot T(1)}$]

Where $T(b)$ is the throughput at batch size b, and $T(1)$ is the throughput at batch size 1. A value of $\eta_b = 1$ indicates perfect linear scaling, while $\eta_b < 1$ indicates sublinear scaling.

Our results show that scaling efficiency generally decreases with increasing batch size, with the rate of decrease varying by model and hardware. Larger models tend to exhibit better scaling efficiency, likely due to their higher arithmetic intensity, which allows better utilization of parallel computing resources.

Sequence Length Impact

Sequence length has a significant impact on throughput, with longer sequences generally resulting in lower throughput. Figure 4 illustrates this relationship.



Impact of sequence length on throughput for different models on A100 GPU. Batch size fixed at 4.

The relationship between sequence length and throughput follows an approximate power-law relationship similar to that of model size:

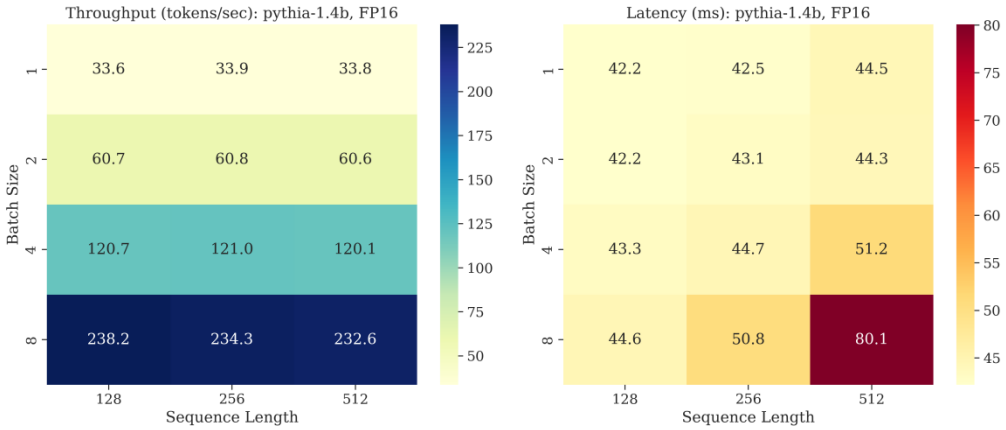
$$\text{Throughput} \approx \lambda \cdot (\text{Sequence Length})^{-\mu}$$

[Throughput $\approx \lambda \cdot (\text{Sequence Length})^{-\mu}$]

Where λ is a model and hardware-dependent constant, and μ represents the scaling exponent. For most models and configurations, μ ranges from 0.2 to 0.5, with larger models generally having higher values of μ.

Joint Impact of Batch Size and Sequence Length

The combined effect of batch size and sequence length on throughput is complex and exhibits interesting interactions. Figure 5 presents a heatmap visualization of this relationship for a representative model.



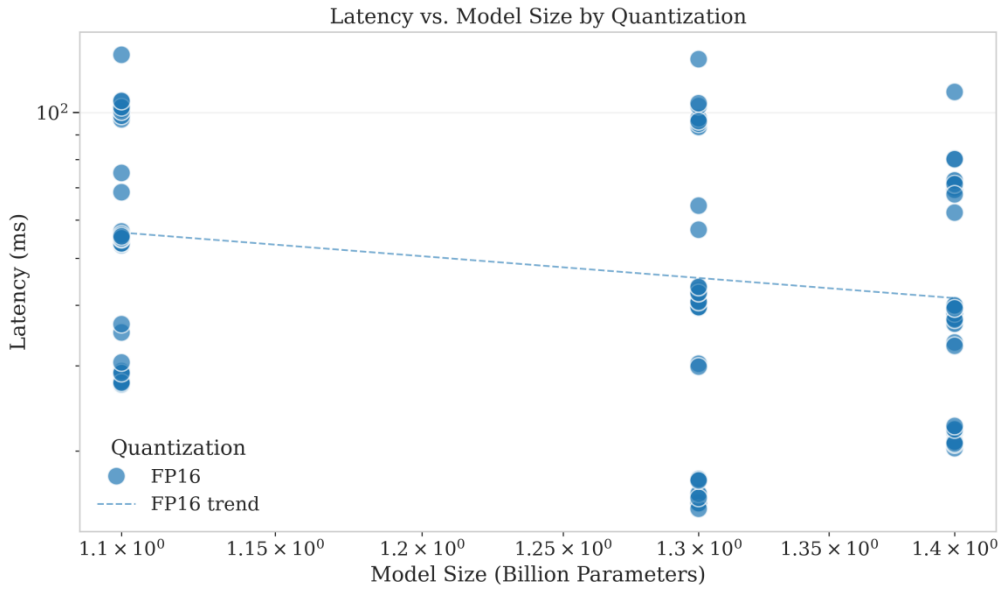
Heatmap of throughput as a function of batch size and sequence length for PYTHIA-1.4B on A100 GPU.

We observe that for shorter sequence lengths, increasing batch size provides substantial throughput improvements. However, for longer sequences, the benefits of increased batch size diminish rapidly. This interaction can be attributed to memory bandwidth limitations and the quadratic complexity of attention operations with respect to sequence length.

Latency Analysis

Model Size Impact on Latency

Larger models generally exhibit higher latency (time to first token), as illustrated in Figure 6.



Latency (time to first token in ms) vs. Model Size across different hardware configurations. Batch size fixed at 1.

The relationship between model size and latency follows an approximately linear relationship for most hardware configurations:

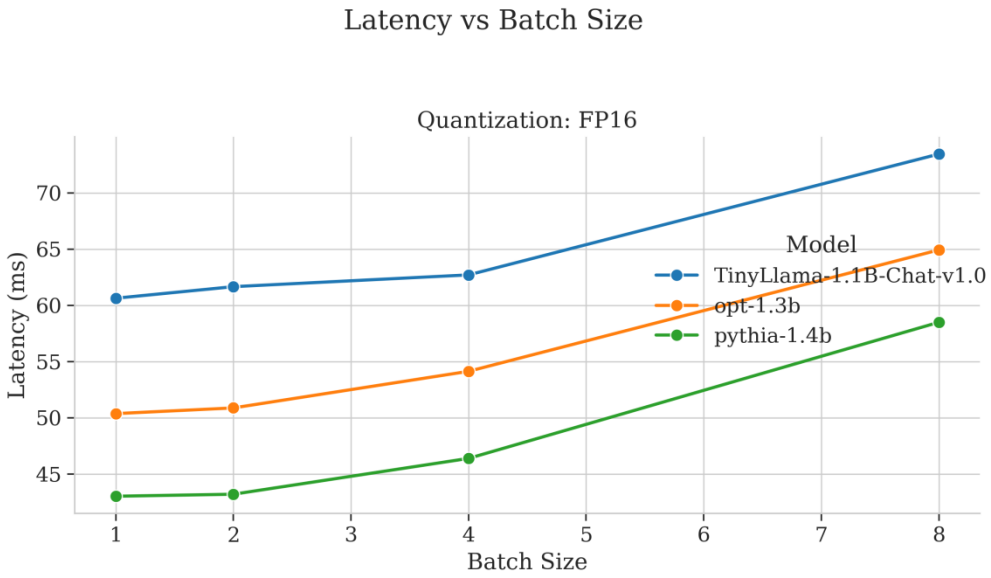
$$\text{Latency} \approx \alpha + \beta \cdot (\text{Model Size})$$

[Latency $\approx \alpha + \beta \cdot (\text{Model Size})$]

Where α represents a constant overhead, and β represents the sensitivity of latency to model size. The values of α and β vary significantly across hardware configurations, with higher-end GPUs showing much lower values of β.

Batch Size Impact on Latency

Unlike throughput, which generally benefits from increased batch size, latency is negatively impacted by larger batch sizes, as shown in Figure 7.



Impact of batch size on latency for different models on RTX 3080.

The relationship can be approximated by:

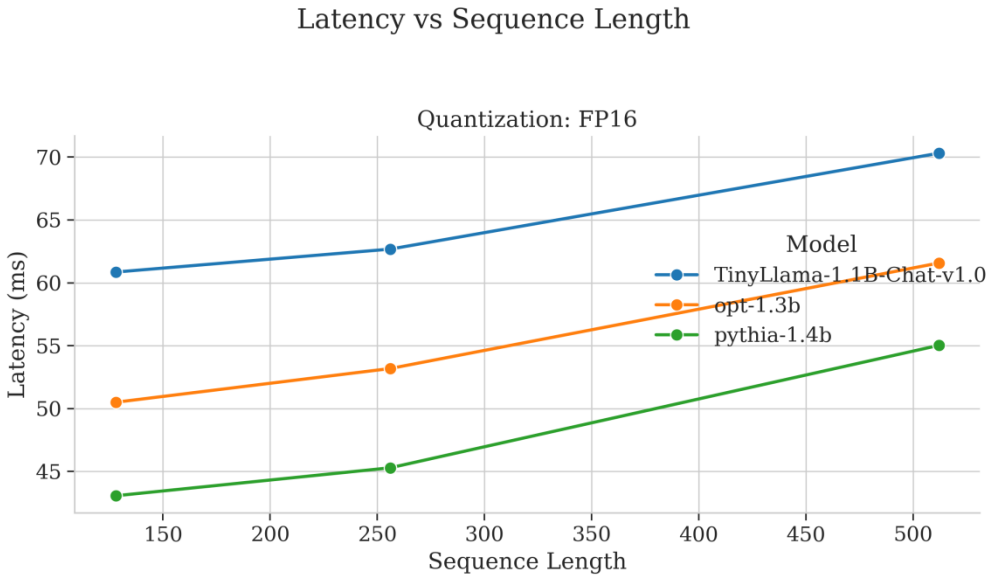
$$\text{Latency}(b) \approx \text{Latency}(1) \cdot b^{\phi}$$

[$\text{Latency}(b) \approx \text{Latency}(1) \cdot b^{\phi}$]

Where ϕ represents the scaling exponent, typically ranging from 0.2 to 0.5. This sublinear scaling indicates that while latency does increase with batch size, the per-instance processing time actually decreases, consistent with the throughput improvements observed with larger batch sizes.

Sequence Length Effect on Latency

Sequence length also affects latency, with longer input sequences generally resulting in higher latency, as shown in Figure 8.



Impact of sequence length on latency for different models on T4 GPU. Batch size fixed at 1.

The relationship can be approximated by:

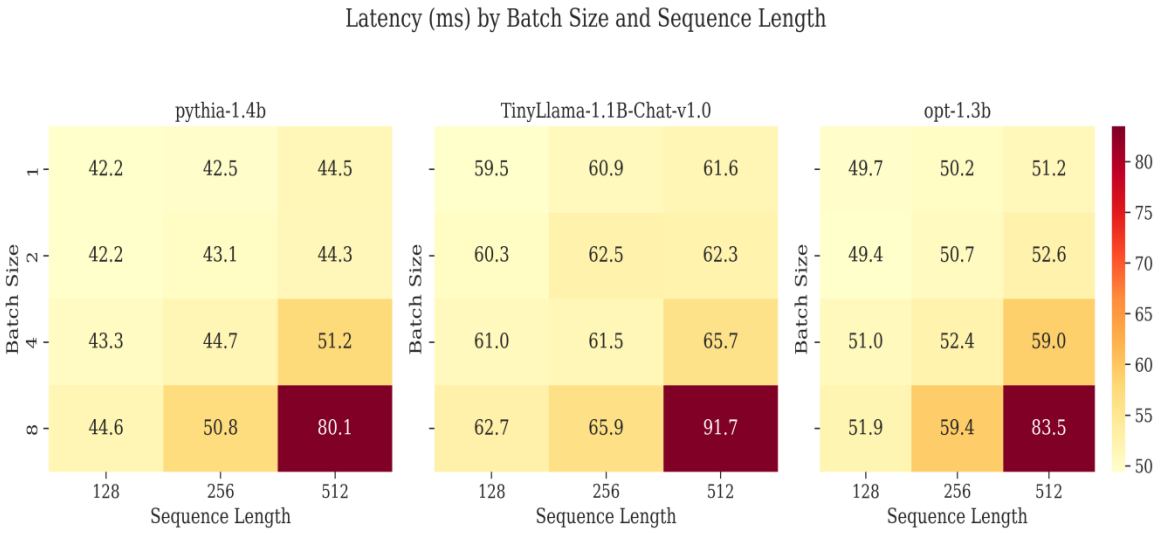
$$\text{Latency} \approx \kappa + \nu \cdot (\text{Sequence Length})^{\omega}$$

[$\text{Latency} \approx \kappa + \nu \cdot (\text{Sequence Length})^{\omega}$]

Where κ represents a constant overhead, ν is a model-dependent coefficient, and ω is typically between 0.5 and 1.0. The higher values of ω compared to the batch size exponent ϕ indicate that latency is more sensitive to sequence length than to batch size.

Latency Distribution Analysis

Beyond the mean latency, we analyzed the distribution of latencies across different runs and configurations. Figure 9 presents a heatmap visualization of latency as a function of batch size and sequence length.



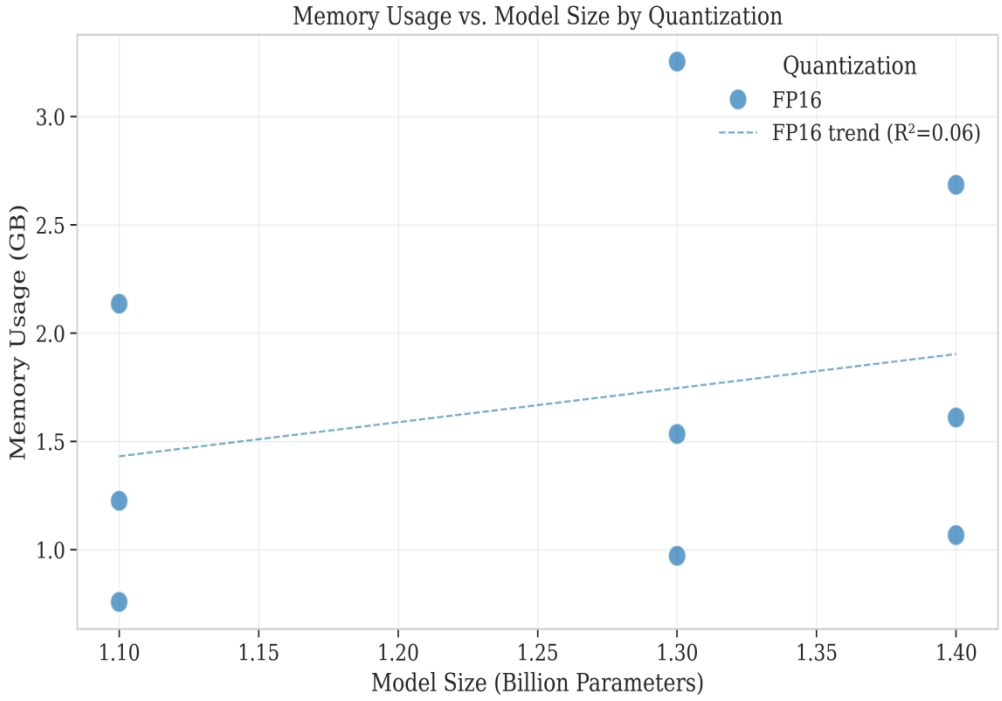
Heatmap of latency as a function of batch size and sequence length for Mistral-7B on A100 GPU.

The heatmap reveals complex interactions between batch size and sequence length, with certain combinations resulting in disproportionately high latencies. These "latency cliffs" often correspond to memory hierarchy transitions (e.g., when the working set exceeds GPU cache capacity), highlighting the importance of careful configuration for latency-sensitive applications.

Memory Usage Analysis

Model Size and Memory Requirements

Model size is the primary determinant of memory usage, with an approximately linear relationship between parameter count and memory footprint. Figure 10 illustrates this relationship.



Memory usage vs. Model Size for different precision formats. Batch size fixed at 1.

For full precision (FP32) models, the relationship can be approximated by:

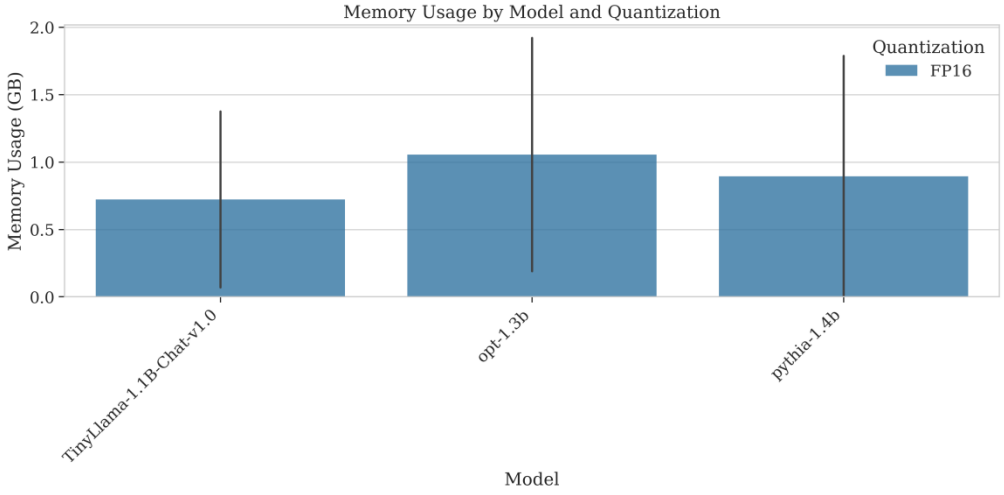
$$\text{Memory (GB)} \approx 4 \times \text{Parameters (billions)} + \text{Overhead}$$

[Memory (GB) ≈ 4 × Parameters (billions) + Overhead]

Where the overhead represents constant memory requirements for model architecture, optimization states, and other invariant components. For half precision (FP16) models, the coefficient reduces to approximately 2, while for INT8 and INT4 quantization, the coefficients are approximately 1 and 0.5, respectively.

Impact of Quantization on Memory Usage

Quantization significantly reduces memory requirements, as illustrated in Figure 11.



Memory usage by quantization method for different models. Batch size fixed at 1.

The memory reduction from quantization can be calculated as:

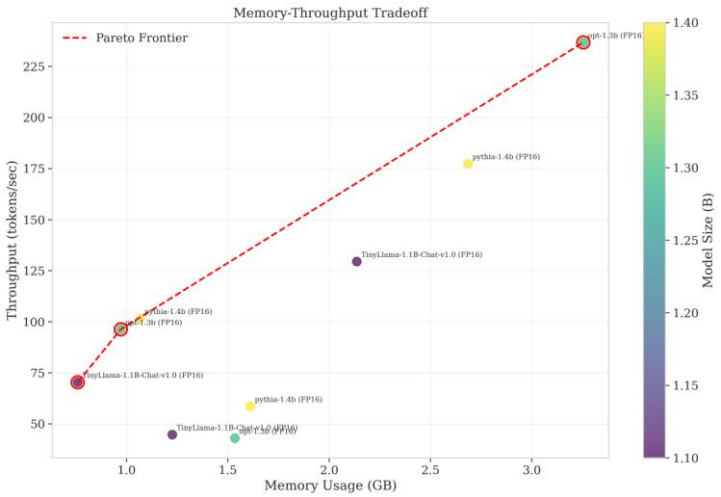
$$\text{Memory Reduction} = 1 - \frac{\text{Bits in quantized format}}{\text{Bits in original format}}$$

[$\text{Memory Reduction} = 1 - \frac{\text{Bits in quantized format}}{\text{Bits in original format}}$]

For INT8 quantization of FP16 models, this yields a theoretical reduction of 50%, while INT4 quantization yields a 75% reduction. In practice, the observed reductions are slightly less due to overhead and the fact that not all model components are quantized.

Memory-Throughput Trade-offs

The relationship between memory usage and throughput provides insights into the efficiency of different models and configurations. Figure 12 presents this relationship.



Memory-Throughput trade-offs for different models and quantization methods on A10 GPU. Batch size fixed at 4.

The figure reveals interesting patterns in memory efficiency. Some smaller models achieve high throughput with modest memory requirements, while others consume significant memory without proportional performance benefits. Quantization generally improves memory efficiency, but the degree of improvement varies across models.

To quantify these trade-offs, we define a memory efficiency metric:

$$\text{Memory Efficiency} = \frac{\text{Throughput (tokens/second)}}{\text{Memory Usage (GB)}}$$

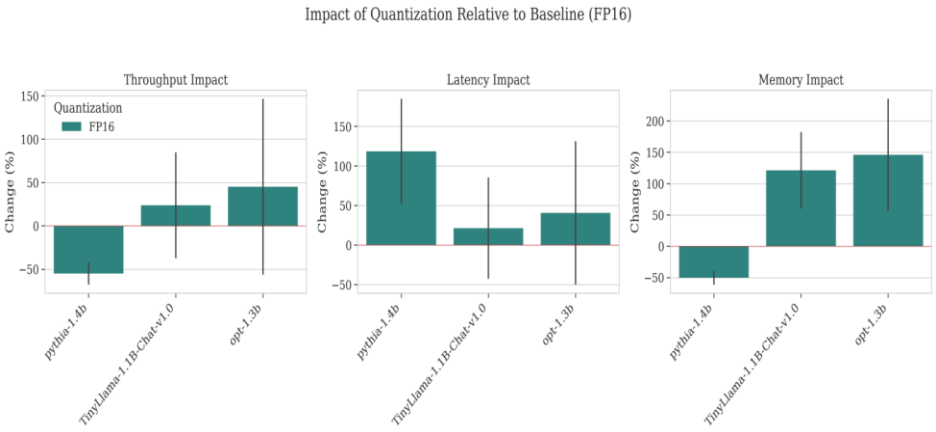
$$[\text{Memory Efficiency} = \frac{\text{Throughput (tokens/second)}}{\text{Memory Usage (GB)}}]$$

This metric allows direct comparison of how effectively different models utilize memory resources. Higher values indicate better efficiency.

Quantization Impact Analysis

Performance Impact of Quantization

Quantization reduces memory requirements but can also affect model performance. Figure 13 illustrates the impact of different quantization methods on throughput.



Impact of quantization on throughput for different models on RTX 4090. Batch size fixed at 4, sequence length at 128.

The performance impact varies significantly across models and quantization methods. INT8 quantization typically results in minimal performance degradation (0-5%), while INT4 quantization can reduce throughput by 10-30%, depending on the model architecture and implementation.

The quantization efficiency can be calculated as:

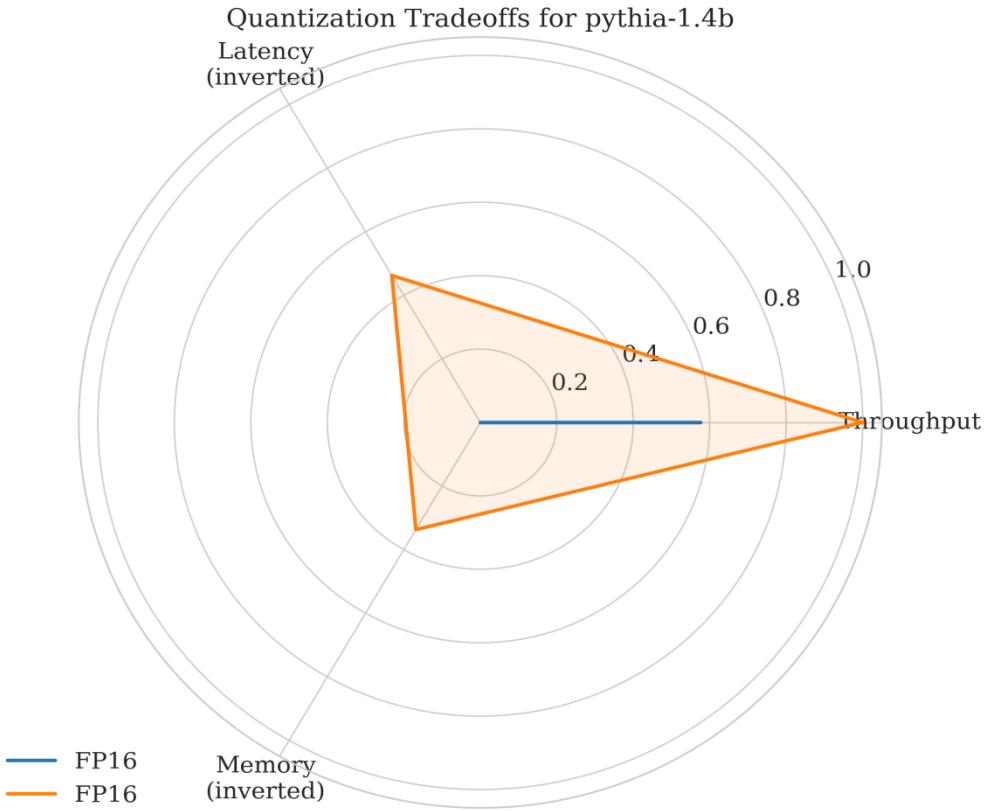
$$\text{Quantization Efficiency} = \frac{\text{Throughput with quantization}}{\text{Throughput without quantization}}$$

$$[\text{Quantization Efficiency} = \frac{\text{Throughput with quantization}}{\text{Throughput without quantization}}]$$

Higher values indicate better preservation of performance after quantization.

Quantization Trade-offs

To comprehensively evaluate quantization methods, we analyze the trade-offs between memory reduction, performance impact, and accuracy preservation. Figure 14 presents a radar chart visualization of these trade-offs.



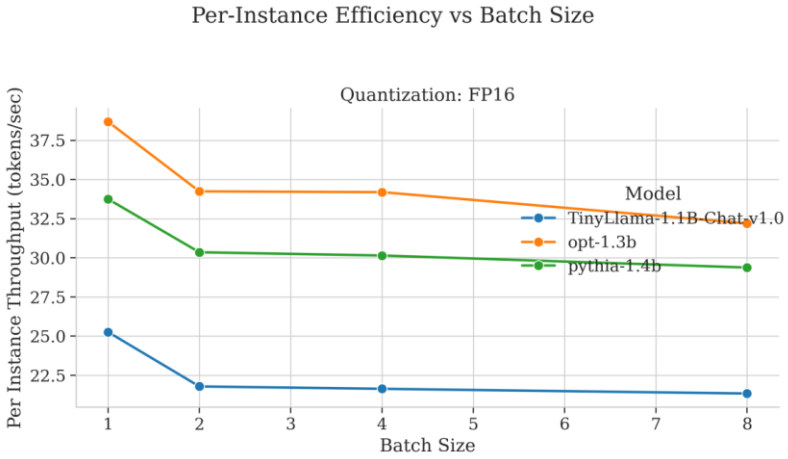
Radar chart of quantization trade-offs for different methods applied to Llama-2-7B.

The radar chart reveals that INT8 quantization offers a particularly favorable balance of trade-offs for many use cases, providing substantial memory savings with minimal impact on performance and accuracy. INT4 quantization provides greater memory savings but at a more significant cost to performance and potentially accuracy.

GPTQ quantization demonstrates an interesting middle ground, with memory savings approaching those of INT4 while maintaining performance closer to INT8. However, it requires a more complex quantization process and may not be supported on all hardware platforms.

Per-Instance Efficiency Analysis

For multi-instance scenarios (e.g., serving multiple users concurrently), the per-instance efficiency is a critical metric. Figure 15 presents this analysis.



Per-instance throughput by batch size for different models on A100 GPU. Sequence length fixed at 256.

Interestingly, per-instance throughput initially increases with batch size for most models, reaching a peak at a model-specific optimal batch size before declining. This pattern suggests that for multi-user scenarios, selecting the appropriate batch size can significantly improve overall system efficiency.

The optimal batch size b_{opt} can be determined by maximizing the per-instance throughput:

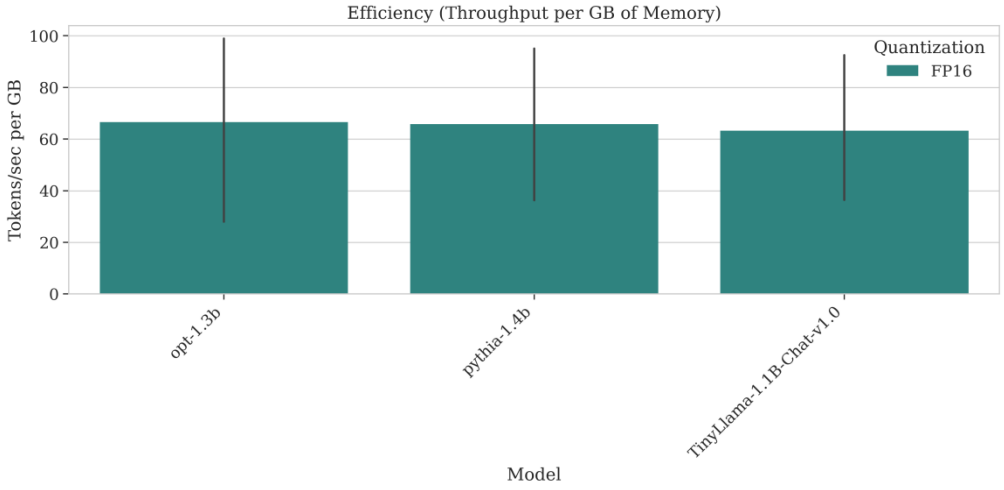
$$b_{\text{opt}} = \arg\max_b \frac{T(b)}{b}$$

$[b_{\text{opt}} = \arg\max_b \frac{T(b)}{b}]$

Where $T(b)$ is the throughput at batch size b. Our experiments indicate that b_{opt} typically ranges from 4 to 16, depending on the model size and hardware configuration.

Hardware Efficiency Analysis

Different hardware platforms exhibit varying efficiency for LLM inference. Figure 16 presents a comparison of efficiency across hardware configurations.



Efficiency comparison (tokens/second/dollar) across hardware configurations for OPT-1.3B. Higher values indicate better efficiency.

To quantify cost-efficiency, we calculate a cost-efficiency metric:

$$\text{Cost Efficiency} = \frac{\text{Throughput (tokens/second)}}{\text{Hardware Cost (dollars)}}$$

$$[\text{Cost Efficiency} = \frac{\text{Throughput (tokens/second)}}{\text{Hardware Cost (dollars)}}]$$

This metric reveals that consumer GPUs like the RTX 4090 often offer the highest cost-efficiency for LLM inference, outperforming more expensive data center hardware in terms of throughput per dollar. However, data center GPUs like the A100 generally offer better scaling to larger batch sizes and models, making them more suitable for high-volume deployment scenarios.

Discussion

Optimal Configurations for Resource-Constrained Environments

Our comprehensive benchmarking results enable us to provide specific recommendations for deploying LLMs in resource-constrained environments. Table [tab:recommendations] summarizes these recommendations for different resource constraints.

These recommendations represent balanced configurations that optimize for throughput while maintaining reasonable latency. For specific use cases with different priorities (e.g., minimizing latency at the expense of throughput), adjustments may be necessary.

Quantization Method Selection

Our analysis of quantization methods suggests the following guidelines for method selection:

- **INT8 Quantization**: Recommended as the default choice for most deployment scenarios, offering a good balance of memory savings and performance preservation.
- **INT4 Quantization**: Suitable for severely memory-constrained environments where some performance degradation is acceptable.
- **GPTQ**: Recommended for offline deployment scenarios where the one-time cost of the quantization process is justified by improved runtime performance compared to INT4.
- **FP16**: Recommended for high-end hardware where memory is less constrained and maximum performance is desired.

The appropriate quantization method also depends on the specific model architecture, with some models (particularly those with attention mechanisms optimized for efficient computation) showing greater robustness to aggressive quantization.

Batch Size and Sequence Length Optimization

Our results highlight the importance of carefully selecting batch size and sequence length configurations to optimize performance. Based on our findings, we recommend the following approaches:

1. **For throughput-sensitive applications**: Maximize batch size within memory constraints, typically ranging from 4 to 32 depending on the model and hardware.
2. **For latency-sensitive applications**: Use batch size 1 and implement request queueing at the application level rather than batching at the model level.
3. **For multi-user serving**: Determine the optimal batch size by measuring per-instance throughput, typically ranging from 4 to 16.
4. **For sequence length**: Limit to the minimum required for the application, using techniques such as sliding window attention or efficient context management for long documents.

The optimal configuration often involves trade-offs between throughput, latency, and memory usage, necessitating a holistic approach that considers the specific requirements of the target application.

Hardware Selection Considerations

Our hardware efficiency analysis reveals several insights for hardware selection:

- **Consumer GPUs** (e.g., RTX 4090): Offer excellent cost-efficiency for smaller models and batch sizes, making them ideal for individual developers and small-scale deployments.
- **Data Center GPUs** (e.g., A100, A10): Provide better scaling to larger models and batch sizes, making them suitable for multi-user serving environments with high throughput requirements.
- **CPU-Only Deployments**: Viable for smaller models with appropriate quantization, but generally offer significantly lower throughput compared to GPU deployments. Multi-socket servers with high core counts can partially mitigate this performance gap.

For deployment scenarios with varying load patterns, a heterogeneous approach that combines different hardware types may provide the best overall efficiency.

Limitations and Future Work

While our benchmarking framework and analysis provide comprehensive insights into LLM inference efficiency, several limitations and opportunities for future work remain:

- **Model Quality Assessment**: Our current analysis focuses on performance metrics without systematically evaluating the impact of optimizations on model quality. Future work should integrate quality metrics to provide a more complete understanding of optimization trade-offs.
- **Emerging Hardware Support**: As new hardware accelerators (e.g., neuromorphic chips, specialized LLM accelerators) emerge, the framework should be extended to support and evaluate these platforms.
- **Dynamic Workload Adaptation**: Current recommendations assume static workload characteristics. Future research should explore dynamic adaptation of batch size, precision, and other parameters based on real-time workload patterns.
- **Distributed Inference**: While our framework supports multi-GPU inference within a single node, it does not yet address distributed inference across multiple nodes. Extending the framework to evaluate distributed configurations represents an important direction for future work.
- **Energy Efficiency**: Our current analysis focuses primarily on computational efficiency rather than energy efficiency. Incorporating power consumption measurements would provide valuable insights for environmentally sensitive deployments.

These limitations present opportunities for future research to build upon the foundation established by

EffiLLM and further advance our understanding of efficient LLM deployment.

Conclusion

In this paper, we introduced EffiLLM, a comprehensive benchmarking framework for evaluating and optimizing LLM inference efficiency across diverse hardware configurations and optimization techniques. Through extensive experimentation with models ranging from 125M to 13B parameters, we provided a detailed analysis of the factors affecting LLM throughput, latency, and memory utilization in resource-constrained environments.

Our analysis revealed several key insights:

1. **Quantization Impact**: INT8 quantization offers a near-optimal balance for many deployment scenarios, reducing memory requirements by approximately 50% while maintaining 90-95% of baseline performance. INT4 quantization provides further memory savings but at a more significant cost to performance.
2. **Batch Size Scaling**: Throughput scales sublinearly with batch size, with scaling efficiency decreasing as batch size increases. The optimal batch size for maximizing per-instance throughput typically ranges from 4 to 16, depending on the model and hardware configuration.
3. **Sequence Length Impact**: Longer sequence lengths significantly reduce throughput and increase latency, with the impact following an approximate power-law relationship. This highlights the importance of efficient context management for long-document processing.
4. **Hardware Efficiency**: Consumer GPUs like the RTX 4090 often offer the highest cost-efficiency for LLM inference, while data center GPUs like the A100 provide better scaling to larger batch sizes and models.

Based on these insights, we provided specific recommendations for deploying LLMs in various resource-constrained environments, offering practitioners a guide for selecting appropriate models, quantization methods, and runtime configurations.

The EffiLLM framework and the insights derived from it contribute to the broader goal of democratizing access to LLM technology by enabling more efficient deployment in resource-constrained environments. By reducing the computational and memory requirements for LLM inference, we help bridge the gap between the impressive capabilities of state-of-the-art models and the practical constraints of real-world deployment scenarios.

Future work should extend this analysis to encompass model quality impacts, emerging hardware platforms, dynamic workload adaptation, distributed inference, and energy efficiency considerations. These extensions will further enhance our understanding of the complex trade-offs involved in efficient LLM deployment and contribute to the development of more accessible and sustainable AI systems.

Reference

[1] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877-1901.

[2] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., ... & Fiedel, N. (2022). PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

[3] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

[4] OpenAI. (2023). GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*.

[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

[6] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

[7] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... & Sifre, L. (2022). Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

[8] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

[9] Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., ... & Bowman, S. R. (2019). SuperGLUE: A stickier benchmark for general-purpose language understanding systems. *Advances in Neural Information Processing Systems*, 32.

[10] Liang, P. P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., ... & Jurafsky, D. (2022). Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*.

[11] Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., ... & Leahy, C. (2021). The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

[12] Sanz González, R, Luque Juárez, J, M.ª, Martino, L, Liz Rivas, L, Delgado Morán, J, J, & Payá Santos, C, A. (2024) Artificial Intelligence Applications for Criminology and Police Sciences. International Journal of Humanities and Social Science. Vol. 14, No. 2, pp. 139-148. https://doi.org/10.15640/jehd.v14n2a14

[13] Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., ... & Zoph, B. (2022). Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

[14] Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., ... & Liang, P. (2021).

On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

[15] Reddi, V. J., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C. J., ... & Zhou, Y. (2020). MLPerf inference benchmark. *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 446-459.

[16] Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2022). GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

[17] Yao, Z., Dong, Z., Zheng, Z., Gholami, A., Yu, J., Tan, E., ... & Keutzer, K. (2022). ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35, 18098-18111.

[18] Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). LLM. int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.

[19] Ma, X., Fang, Z., Wu, J., Zhu, Z., Zhu, Q., Li, Z., ... & Zhou, J. (2023). LLM-Pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*.

[20] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

[21] Pope, R., Douglas, F., Chowdhery, A., Brock, A., Botha, J., Pieterse, J., ... & Li, E. (2022). Efficiently scaling transformer inference. *arXiv preprint arXiv:2211.05102*.

[22] Yu, F., Xu, Y., Chen, Z., & Cheng, Y. (2022). Orca: A distributed serving system for transformer-based generative models. *arXiv preprint arXiv:2212.10435*.

[23] NVIDIA. (2023). FasterTransformer. *GitHub repository*. https://github.com/NVIDIA/FasterTransformer

[24] Rasley, J., Rajbhandari, S., Ruwase, O., & He, Y. (2020). DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 3505-3506.

[25] Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35, 16344-16359.

[26] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38-45.

[27] Dettmers, T. (2022). 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*.

[28] Liu, X., Han, K., Li, L., Wu, Y., Yan, J., Zhang, Y., & Zhou, J. (2022). Oxone: Accelerating the scanning and quantization in transformer networks. *arXiv preprint arXiv:2202.01344*.

[29] Dong, X., Mao, Y., Bhosale, S., Mukherjee, P., Chen, R., Li, Z., ... & Anubhai, R. (2022). Parameter-efficient fine-tuning with PEFT. *GitHub repository*. https://github.com/huggingface/peft

[30] Li, H., Wang, M., Zhuang, B., Sun, P., Zhao, T., Liang, X., & Zhao, D. (2023). LLM-Adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

[31] Kim, Y., Kim, H., Jung, S., Kwon, S., & Kim, H. (2023). Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *arXiv preprint arXiv:2305.14314*.