Optimizing LLM Inference for Headline Generation

Author: Survesh
Date: October 2025
Environment: AWS SageMaker Notebook (multi-GPU), Python 3.10, PyTorch 2.1, Transformers ≥4.38

Objective

To optimize inference performance of Meta's Llama 3 models for the task of headline generation from article summaries. The goal is to reduce latency and memory usage while preserving output quality, measured via ROUGE scores.

Methodology

Task Setup

- **Input:** News article summaries
- **Output:** Concise headlines
- **Prompt Engineering:** Generate a concise headline for the following news summary: {summary}
- **Evaluation Metric:** ROUGE-1, ROUGE-2, ROUGE-L, ROUGE-Lsum (via evaluate library)

Libraries Used

- transformers (Hugging Face): model loading, generation
- accelerate: device mapping for parallelism
- bitsandbytes: 8-bit and 4-bit quantization
- torch: pruning, timing, memory profiling
- evaluate: ROUGE scoring
- time: latency measurement

Optimization Techniques Applied

1. Baseline Inference
- Model: meta-llama/Meta-Llama-3-1B
- use_cache=False

2. KV-Caching
- Enabled use_cache=True
- Reuses attention keys/values across steps

3. Pruning
- Global unstructured L1 pruning on linear layers
- Reduced parameter count

4. Quantization
- 8-bit and 4-bit loading via bitsandbytes
- Lower memory footprint

5. Distributed Inference
- **Tensor Parallelism:** device_map="auto" across GPUs
- **Pipeline Parallelism:** device_map="balanced"

6. Speculative Decoding
- Draft: Meta-Llama-3-1B, Target: Meta-Llama-3-7B
- Used assistant_model and num_prediction=4

7. Target Model Only
- Full-size Meta-Llama-3-7B without speculative decoding

Final Benchmark Results

Optimization Technique	Avg Latency (s)	P99 Latency (s)	Throughput (samples/sec)	Max GPU Memory (MB)	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-Lsum
Baseline	1.754	2.098	0.570	2372.75	0.1973	0.0712	0.1730	0.1740
KV Caching	0.712	0.744	1.404	2372.75	0.1964	0.0732	0.1746	0.1751
Magnitude Unstructured Pruning	0.599	0.625	1.670	10868.62	0.1458	0.0328	0.1306	0.1309
Quantized 8-bit Model	1.807	1.960	0.553	10868.62	0.2000	0.0764	0.1697	0.1712
Quantized 4-bit Model	0.914	0.931	1.095	10868.62	0.1526	0.0412	0.1347	0.1338
Tensor Parallel	0.720	0.825	1.389	10868.62	0.1959	0.0732	0.1731	0.1761
Pipeline Parallel	0.719	0.729	1.390	10868.62	0.1959	0.0732	0.1731	0.1761
Speculative Decoding	2.416	5.425	0.414	13247.43	0.1359	0.0322	0.1060	0.1104
Target Model Only	1.715	2.041	0.583	13247.43	0.2011	0.0757	0.1786	0.1802

Trade-off Analysis

0. Baseline (Llama-3-1B, no KV-caching)

- Performance: Moderate latency and low throughput. Acts as the control for all other techniques.
- Memory: Efficient and fits comfortably on a single GPU.
- Quality: ROUGE scores are decent, showing that the model can generate reasonable headlines without optimization.
- Summary: A reliable starting point. Good balance of quality and resource usage, but slower than optimized variants.

1. KV-Caching

- Performance: Dramatically reduces latency by avoiding redundant computation during autoregressive decoding. The model reuses previously computed attention keys and values, which speeds up generation without changing the output.
- Memory: No additional memory savings, but no increase either it's a purely computational optimization.
- Quality: ROUGE scores remain stable or slightly improve due to faster decoding and reduced token delay.
- Summary: A low-risk, high-reward optimization. Ideal for any deployment scenario where latency matters.

2. Magnitude Unstructured Pruning

- Performance: Improves latency and throughput by removing low-magnitude weights from linear layers. This reduces the number of active parameters and speeds up matrix operations.
- Memory: Significant reduction in memory usage due to fewer active weights.
- Quality: ROUGE scores drop sharply, indicating that pruning harms headline generation quality.
- Summary: Best suited for extreme resource-constrained environments where speed is critical and quality can be sacrificed. Not recommended for production headline generation.

3. Quantized 8-bit Model

- Performance: Slightly slower than baseline due to quantization overhead, especially on CPUs or non-optimized GPU kernels.
- Memory: Major memory savings model weights occupy half the space of full precision.
- Quality: ROUGE scores are stable or slightly better than baseline
- Summary: Excellent choice for deployment on limited hardware. Should be combine with KV-caching for best results.

4. Quantized 4-bit Model

- Performance: Faster than 8-bit and baseline due to smaller weight matrices and reduced memory bandwidth.
- Memory: Extreme memory savings ideal for edge devices or low-cost GPU setups.
- Quality: ROUGE scores drop moderately, indicating some loss of semantic richness and fluency.
- Summary: A strong option for low-memory environments, but not ideal if headline quality is critical.

5. Tensor Parallelism

- Performance: Improves throughput and latency by splitting tensor operations across multiple GPUs. Each GPU handles part of the computation in parallel.
- Memory: Efficient memory distribution across devices, allowing larger models to run without bottlenecks.
- Quality: ROUGE scores remain stable, as the model architecture and weights are unchanged.
- Summary: Ideal for multi-GPU setups. Offers strong performance gains without compromising quality.

6. Pipeline Parallelism

- Performance: Similar to tensor parallelism, but splits model layers sequentially across GPUs. Each GPU handles a stage of the forward pass.
- Memory: Enables deep models to run across devices, reducing per-GPU memory load.
- Quality: Identical to tensor parallelism no change in output quality.
- Summary: Best for very large models (e.g., 13B+) where layer depth exceeds single-GPU capacity. Slightly more complex to manage than tensor parallelism.

7. Speculative Decoding

- Performance: Surprisingly slower in this setup due to mismatch between draft and target models. Latency and P99 latency are higher than baseline.
- Memory: Requires both models in memory, leading to the highest GPU usage in the benchmark.
- Quality: ROUGE scores are significantly lower, suggesting that the draft model's predictions are frequently rejected or misaligned.
- Summary: Promising technique in theory, but underperformed here due to poor draft-target synergy. May work better with smaller gaps between models or fine-tuned drafts.

8. Target Model Only (Llama-3-7B)

- Performance: Slower than baseline due to larger model size and deeper architecture.
- Memory: Highest memory usage in the benchmark.
- Quality: Best ROUGE scores across all variants headlines are more fluent, informative, and accurate.
- Summary: Ideal for quality-first applications. Use with parallelism or quantization to mitigate resource demands.

Technique	Quality Impact	Notes
KV-Caching	⬆️ Slight gain	Best latency boost with no quality loss
Pruning	⬇️ Significant drop	Fast but poor headline quality
Quantized 8-bit	⬆️ Slight gain	Good for deployment, slower than baseline
Quantized 4-bit	⬇️ Moderate drop	Best for ultra-low memory environments
Tensor/Pipeline Parallelism	⬆️ Stable	Ideal for multi-GPU setups
Speculative Decoding	⬇️ Significant drop	High latency and low quality in this setup
Target Model Only	⬆️ Best quality	Slowest but highest ROUGE scores

Final Recommendations

Scenario	Recommended Strategy
Low-latency deployment	KV-Caching
Memory-constrained hardware	Quantized 4-bit + KV-Caching
Multi-GPU environment	Tensor or Pipeline Parallelism
Highest quality headlines	Target Model Only (7B)
Balanced quality/speed	KV-Caching + Quantized 8-bit

Conclusion

This project demonstrates that

- LLM inference can be significantly optimized using a combination of architectural, compression, and distributed strategies.
- KV-Caching offers the best latency-quality trade-off,
- Quantization and parallelism unlock scalability.
- Speculative decoding underperformed in this context due to draft-target mismatch but remains promising with better-aligned models.