A100 Optimization for gpt-oss-120b: A Comprehensive Analysis of Architecture, Performance, and Economics

1. Executive Summary

The deployment of large-scale Mixture-of-Experts (MoE) models, particularly those exceeding the 100-billion parameter threshold, has traditionally necessitated the use of the latest generation of accelerator hardware, primarily the NVIDIA H100 "Hopper" architecture. The release of gpt-oss-120b by OpenAI, utilizing the MXFP4 (Microscaling Format 4-bit) quantization standard, challenges this paradigm. This report provides an exhaustive technical analysis of deploying gpt-oss-120b on the previous-generation NVIDIA A100-80GB GPU using vLLM v0.10.2.

Our research confirms that the A100-80GB is not only a viable platform for this workload but, when rigorously optimized, offers a price-performance ratio that frequently surpasses that of H100-based solutions in specific operational regimes. The key enabler of this capability is the **Marlin kernel**, a specialized software-hardware co-design that allows the Ampere architecture—which lacks native FP4 hardware support—to emulate 4-bit execution through a sophisticated "Just-In-Time" (JIT) dequantization pipeline within the GPU's shared memory. This mechanism reduces the memory footprint of the model weights to approximately 61 GB, fitting comfortably within the A100's 80 GB envelope while maintaining computational throughput competitive with native FP16 execution in memory-bound scenarios.[1]

Performance benchmarking reveals that a single A100 can sustain system-level throughputs of roughly 4,700 to 5,000 tokens per second (TPS) under optimal batching conditions, with short-context bursts exceeding 10,000 TPS.[4] However, achieving these metrics requires navigating a complex landscape of configuration pitfalls. We identify the default FLASH_ATTN backend and **Prefix Caching** features in vLLM v0.10.2 as critical stability risks. Specifically, prefix caching is prone to hash collision bugs that result in output corruption and severe latency degradation, necessitating its disablement in production environments until the release of patches in v0.11.1+.[5]

This report delineates a "Golden Configuration" for vLLM v0.10.2 that balances the aggressive memory savings of MXFP4 with the rigorous stability requirements of enterprise inference. Furthermore, our economic analysis demonstrates that while the H100 offers approximately double the raw throughput, the A100's lower spot market cost (often 40-50% cheaper) makes it the superior choice for workloads that do not consistently saturate the massive

throughput capacity of Hopper GPUs.[7]

---

2. Introduction: The Democratization of Hyperscale MoE Models

The landscape of Large Language Model (LLM) inference is undergoing a structural shift, driven by the dual pressures of model scale and operational cost. The emergence of gpt-oss-120b, an open-weights reasoning model employing a Mixture-of-Experts (MoE) architecture, represents a watershed moment. Unlike dense models where every parameter is active for every token, MoE models activate only a subset of experts per token (sparse activation), decoupling total parameter count from inference latency. However, the total parameter count still dictates the VRAM requirement, creating a "memory wall" that typically precludes single-GPU deployment for models of this size.

2.1 The 100B+ Parameter Barrier

Historically, a 120-billion parameter model in standard FP16 (16-bit floating point) precision requires roughly 240 GB of GPU memory merely to store the weights. Deployment would traditionally require a cluster of four A100-80GB GPUs (320 GB total) or roughly eight A100-40GB cards to accommodate the model, KV cache, and activation overhead. This hardware requirement imposes a significant barrier to entry, both in terms of capital expenditure (CapEx) and operating expense (OpEx).

The introduction of gpt-oss-120b with native **MXFP4 quantization** radically alters this calculus. By reducing the weight precision to 4 bits using the Microscaling Format (MX), the model's storage footprint is compressed by a factor of four, bringing the weight requirement down to approximately 60 GB.[1] This compression theoretically enables the entire model to fit onto a single A100-80GB card. However, fitting the model into memory is only the first challenge; executing it efficiently on hardware that predates the MXFP4 standard requires a complex orchestration of software kernels and memory management strategies.

2.2 Scope of Analysis

This report focuses on the technical validation and optimization of this single-GPU deployment strategy. We examine the interaction between the vLLM inference engine (specifically version v0.10.2, as identified in the user's environment) and the NVIDIA Ampere architecture. The analysis spans four critical dimensions:

1. **Architectural Viability:** How the Marlin kernel bridges the gap between MXFP4 data and Ampere's FP16 tensor cores.
2. **Performance Optimization:** Tuning batch sizes, attention backends, and memory utilization to maximize throughput.
3. **System Stability:** Mitigating known bugs in vLLM's caching and scheduling subsystems.

4. **Economic Justification:** Evaluating the Total Cost of Ownership (TCO) relative to migrating to newer H100 infrastructure.

---

3. Architectural Foundations: MXFP4 and the Ampere Constraint

To optimize the deployment of gpt-oss-120b on the A100, it is essential to understand the underlying mechanics of the MXFP4 format and how vLLM emulates support for it on the Ampere architecture, which natively lacks the necessary instructions.

3.1 The MXFP4 Microscaling Standard

The Open Compute Project (OCP) Microscaling Format (MX) is designed to address the limitations of standard integer quantization. Traditional INT4 quantization typically applies a single scaling factor to a large tensor or channel. While efficient, this coarse granularity often results in significant precision loss for the "outlier" weights common in large transformer models.

MXFP4 addresses this by introducing a hierarchical scaling structure. Weights are grouped into small blocks (e.g., 32 elements), and each block shares a common scaling factor. This fine-grained quantization preserves the dynamic range of the model weights, allowing the 120B parameter model to retain its reasoning capabilities despite the aggressive 4-bit compression.

The gpt-oss-120b model is distributed with its MoE weights pre-quantized in this format. The implications for memory bandwidth are profound. In an inference workload, the primary bottleneck is often the speed at which weights can be moved from High Bandwidth Memory (HBM) to the compute units. By compressing weights to 4 bits, the effective bandwidth of the A100 is quadrupled relative to FP16, allowing the GPU to feed its Tensor Cores at a rate that matches or exceeds their computational throughput for memory-bound operations.[1]

3.2 The Marlin Kernel: Bridging the Hardware Gap

A common misconception is that running gpt-oss-120b on an A100 involves native 4-bit computation. The NVIDIA A100 (Compute Capability 8.0) is built on the Ampere architecture, which supports FP16, BF16, TF32, and INT8/INT4 tensor operations, but notably **does not support the specific FP4 or MXFP4 formats** introduced in the Blackwell (Compute Capability 10.0) generation.[10]

To run this model, vLLM utilizes the **Marlin** kernel, a high-performance mixed-precision kernel developed to enable 4-bit inference on Ampere and Ada Lovelace GPUs.[3] Marlin is not a simple dequantization wrapper; it is a fundamental re-architecting of the matrix multiplication

pipeline designed to hide the latency of dequantization.

3.2.1 The Shared Memory Pipeline

The core innovation of Marlin lies in its utilization of the GPU's memory hierarchy. A standard matrix multiplication kernel loads weights from HBM to registers and then computes. Marlin intervenes in this process:

1. **Asynchronous Load (cp.async):** The kernel leverages the cp.async instruction, a feature introduced in the Ampere architecture, to copy compressed 4-bit weights from global memory (HBM) directly into the L1/Shared Memory (SMEM). This bypasses the register file for the initial load, reducing register pressure and latency.[12]
2. **Shared Memory Dequantization:** Once the data is in the high-bandwidth Shared Memory, Marlin performs an optimized "on-the-fly" dequantization. It converts the 4-bit integers into 16-bit floating-point values (FP16 or BF16) using a highly efficient bit-manipulation sequence.
3. **Pipelined Execution:** This dequantization is pipelined with the compute stage. While one warp of threads is performing matrix multiplication on a block of dequantized weights, another warp is pre-fetching and dequantizing the next block.
4. **Tensor Core Dispatch:** The now-FP16 weights are fed into the standard Ampere Tensor Cores. From the perspective of the compute unit, it is performing a standard FP16 operation.

3.2.2 The "Puzzling" Layout

To make this dequantization efficient, the weights cannot be stored in standard row-major or column-major order. Marlin requires the weights to be pre-shuffled into a specific layout that aligns with the GPU's thread access patterns. Snippets indicate that weights are stored interleaved, often following a pattern like 64207531 within a 32-bit integer, to allow for parallel decoding by multiple threads without bank conflicts in shared memory.[12] This pre-processing is handled by vLLM during the model loading phase or offline conversion, but it explains why model loading times can sometimes be longer for Marlin-optimized checkpoints.

3.3 Implications for Optimization

The reliance on Marlin defines the optimization strategy for the A100. Because the dequantization overhead is hidden by the memory fetch latency (which is reduced by 4x due to compression), the kernel shifts the workload's bottleneck profile.

- **Memory Bound $\rightarrow$ Compute Bound:** For small batch sizes, the workload remains memory bound, and Marlin provides a massive speedup (near 4x).
- **Saturation Point:** As batch size increases, the workload eventually becomes compute-bound (limited by the 312 TFLOPS of the Tensor Cores). The A100 needs a large number of concurrent threads to hide the pipeline latency fully. This dictates that **maximizing batch size** (via max_num_seqs and max_num_batched-tokens) is the

primary lever for throughput performance on this hardware.[4]

---

4. The vLLM Execution Engine (v0.10.2) Analysis

The user's environment is fixed to vLLM v0.10.2. This specific version represents a transitional state in the development of vLLM, containing legacy features that are critical for stability on Ampere but which have been deprecated in newer releases.

4.1 Attention Backend: The TRITON_ATTN Imperative

The gpt-oss-120b model employs a specific attention mechanism known as "Sliding Window Attention" with "Attention Sinks." This architectural choice allows the model to handle theoretically infinite input streams by retaining a fixed window of recent tokens plus a few initial tokens (sinks) to maintain stability, while discarding the middle context.

Standard implementations of FlashAttention (specifically FlashAttention-2) available in vLLM v0.10.2 do not natively support this complex masking pattern on the Ampere architecture with full correctness.[15] While FlashAttention-2 supports sliding windows in general, the specific "sink" implementation often requires custom kernel logic.

4.1.1 The Triton Kernel Solution

For v0.10.2, the documentation and user reports explicitly point to TRITON_ATTN_VLLM_V1 as the required backend for gpt-oss on Ampere.[16]

- **Mechanism:** This backend utilizes OpenAI's Triton language to generate a custom CUDA kernel at runtime. This kernel is specifically written to handle the strided memory access patterns of the sliding window and the persistence of the attention sinks.
- **Performance vs. Compatibility:** While pure CUDA kernels (like those in FlashAttention) are often slightly faster due to hand-tuning, the Triton kernel guarantees correctness for this specific model architecture. Attempting to force FLASH_ATTN in this version often results in silent failures where the model "forgets" the system prompt (stored in the sinks) or crashes with invalid memory access errors.[15]

Critical Configuration:

Users must explicitly set the environment variable VLLM_ATTENTION_BACKEND=TRITON_ATTN_VLLM_V1. It is vital to note that in subsequent versions (v0.11.0 and later), this backend was deprecated and its functionality merged into the unified FLASH_ATTN and FLASHINFER backends. Therefore, this configuration is specific to the v0.10.2 constraint and must be revisited if the system is upgraded.

4.2 Asynchronous Scheduling

The gpt-oss-120b model, being an MoE, introduces additional overhead in the scheduling

phase. For every token generated, the model must determine which experts to route data to. While this routing computation happens on the GPU, the management of the request queue and the allocation of PagedAttention blocks happens on the CPU (in Python).

The --async-scheduling flag is identified as a key optimization.[4]

- **Synchronous Mode:** The CPU prepares a batch, launches the GPU kernel, waits for completion, and then prepares the next batch. This serializes CPU and GPU latency.
- **Asynchronous Mode:** The CPU prepares the metadata for batch $N+1$ while the GPU is still executing batch N.
- **Impact:** Benchmarks indicate a throughput improvement of approximately **4.6%** when this feature is enabled.[4] Given the high cost of GPU time, this is a "free" efficiency gain that should always be enabled for production deployments.

4.3 PagedAttention and Memory Management

vLLM's defining feature, PagedAttention, allows the KV cache to be stored in non-contiguous memory blocks, eliminating external fragmentation. For a 120B model on an 80GB card, memory fragmentation is the enemy. A single contiguous allocation requirement could cause an Out-Of-Memory (OOM) error even if gigabytes of free RAM exist in scattered holes. PagedAttention prevents this, enabling the "exceeding estimates" context length of 128k reported by the user.

---

5. Performance Characterization and Benchmarking

To validate the deployment, we analyze the performance metrics derived from benchmark data on A100 hardware.[4] The data reveals a distinct profile that varies significantly based on prompt length and batching intensity.

5.1 Throughput Analysis

Throughput is the primary metric for batch processing and high-concurrency serving. The following table summarizes the system-level throughput for gpt-oss-120b on A100 under optimized conditions (Async Scheduling + Tuned Batching).

Scenario	Input Length	Throughput (Tokens/Sec)	Latency (TPOT)	Optimization Gain
ShareGPT (Mix)	Variable	**5,055 TPS**	94.55 ms	+6.0%

Short Prompt	128 tokens	**10,785 TPS**	301.66 ms	+15.3%
Medium Prompt	2,000 tokens	**9,249 TPS**	117.85 ms	+9.4%
Long Prompt	4,000 tokens	**8,468 TPS**	99.77 ms	+10.4%
Very Long Prompt	32,000 tokens	**4,605 TPS**	201.61 ms	+6.2%

Analysis:

- **The "10k TPS" Burst:** The extremely high throughput for short prompts (10,785 TPS) indicates that the Marlin kernel is incredibly efficient when the workload is dominated by the prefill phase (processing input tokens in parallel). In this regime, the GPU is fully saturated.
- **The "5k TPS" Sustained:** For longer contexts (32k), throughput drops to ~4,600 TPS. This is expected as the memory bandwidth required to load the massive KV cache competes with the bandwidth required to load the model weights.
- **User's "50-100 tok/s":** The user reported "50-100 tok/s." This figure likely refers to the generation speed of a *single* request (per-stream throughput). In a batch of 1, getting 50-100 tok/s implies a latency of 10-20ms per token. However, the benchmark shows mean TPOTs of ~100ms (10 tok/s) under load. This discrepancy highlights the trade-off: **High System Throughput comes at the cost of Individual Latency.** To achieve 5,000 TPS, the system must be processing hundreds of requests concurrently, slowing down each individual stream.

5.2 Latency Characteristics

The Time Per Output Token (TPOT) metrics range from **94ms** to **301ms** in the optimized benchmarks.

- **94ms (ShareGPT):** This is the typical experience for a user in a mixed workload. It translates to roughly 10-11 tokens per second, which is roughly reading speed for humans.
- **301ms (Short Prompt):** This high latency for short prompts in the *Optimized* column (vs 182ms in Baseline) is counter-intuitive but likely reflects the aggressive batching. By forcing a larger batch of short requests, the system increases total throughput (10k TPS) but makes each request wait longer in the queue or compute cycle.

5.3 Batching Strategy

To achieve the benchmarked "Optimized" numbers, specific vLLM parameters are required [4]:

- --max-num-batched-tokens 4096: This parameter limits the number of tokens the scheduler will pack into a single forward pass. A value of 4096 is identified as the "sweet spot" for A100. Lower values leave the GPU underutilized; higher values can cause OOMs or excessive scheduling latency.
- --max-num-seqs: While not explicitly capped in the benchmark snippet, a value of roughly **256** is recommended for 80GB cards to balance VRAM usage with concurrency.

---

6. Parallelism Strategies: Tensor vs. Expert

The user explicitly requests optimization of "MoE parallelism." In the context of vLLM and gpt-oss-120b, this refers to the choice between Tensor Parallelism (TP) and Expert Parallelism (EP) when scaling beyond a single GPU.

6.1 Tensor Parallelism (TP)

Tensor Parallelism splits every weight matrix in the model across multiple GPUs. For every layer, the GPUs compute a partial result and then synchronize via an All-Reduce operation.

- **Pros:** Mature, stable implementation in vLLM. Linearly reduces memory pressure per GPU.
- **Cons:** Requires extremely high bandwidth between GPUs.
- **A100 Suitability:**
  - **With NVLink (SXM4):** TP is highly effective. 2x A100s with NVLink can run the model at TP=2, doubling the effective memory bandwidth and potentially halving latency.
  - **Without NVLink (PCIe):** TP is **catastrophic**. The PCIe bus latency is too high for the frequent synchronization required by TP. Performance will likely be worse than a single GPU.

6.2 Expert Parallelism (EP)

Expert Parallelism assigns whole experts to different GPUs. For a given token, the router identifies the necessary expert and sends the token to the GPU hosting that expert.

- **Pros:** Requires "All-to-All" communication, which scales better than All-Reduce for sparse MoEs because only active tokens are moved, not full tensor gradients/activations.
- **Cons:** Implementation maturity. Snippets explicitly mention that enabling EP (--enable-expert-parallel) for gpt-oss on vLLM results in runtime errors such as "Boolean value of Tensor with more than one value is ambiguous".[18]
- **Status in v0.10.2:** The implementation is marked as **experimental** and is unstable on Ampere hardware for this model.
- **Recommendation: Avoid Expert Parallelism** for production deployments on v0.10.2. Stick to Tensor Parallelism (TP=1 for single card, TP=2/4 for multi-card NVLink setups).

7. Stability and Reliability: The Prefix Caching Crisis

A critical component of the user's request is to "check Prefix caching stability." Our research uncovers a severe, high-impact bug in vLLM v0.10.2 that makes this feature dangerous for production use.

7.1 The Mechanism of Failure

Prefix Caching allows the engine to recognize that a new request shares a common prefix (e.g., a long system prompt) with a previous request. It then reuses the KV cache blocks for that prefix, saving computation. This relies on hashing the prompt to find a match.

In v0.10.2, the hashing mechanism for RadixAttention contains a flaw in how it handles LoRA adapters and certain complex prompt structures. Specifically, it often fails to distinguish between different LoRA requests or complex prompt variations due to hash collisions (e.g., using lora_name instead of a unique lora_int_id as the hash key).[6]

7.2 Symptoms of Corruption

When a collision occurs, the model retrieves the "wrong" KV cache blocks. The attention mechanism effectively attends to context that does not match the current input.

- **Garbage Output:** The model generates gibberish.
- **Infinite Loops:** A hallmark symptom reported is the model getting stuck repeating a specific phrase, such as "addCriterion," indefinitely.[5]
- **Cross-Request Leakage:** In multi-tenant environments, one user's prompt might inadvertently pollute the cache used by another user.
- **Performance Regression:** Paradoxically, enabling the feature has been observed to *increase* latency by up to 2x due to the overhead of managing these corrupted cache states and the resulting generation loops.[5]

7.3 Mitigation Strategy

Immediate Action: Disable Prefix Caching.

The flag --no-enable-prefix-caching must be passed to the serving command. While this sacrifices the potential throughput gain for shared-prompt workloads, the risk of serving corrupted data is unacceptable. This issue is reportedly resolved in vLLM v0.11.1, offering a clear upgrade path, but for the current v0.10.2 deployment, the feature is considered broken.

8. Operational Best Practices and Golden

Configuration

Based on the synthesis of architectural constraints, performance benchmarks, and stability risks, we propose the following "Golden Configuration" for deploying gpt-oss-120b on a single A100 using vLLM v0.10.2.

8.1 Memory Optimization: The FP8 KV Cache

To maximize batch size (and thus throughput) on the memory-constrained A100, minimizing the KV cache footprint is paramount.

- **Setting:** --kv-cache-dtype fp8
- **Justification:** While the A100 cannot compute in FP8, vLLM supports **storing** the KV cache in FP8 (e5m2 or e4m3 format) and converting it to FP16 on the fly for attention calculation. This effectively **doubles the available context window** or batch size compared to the default FP16, with negligible impact on model accuracy.[11]

8.2 Recommended Launch Command

Bash

```
# Set environment for Triton backend correctness on Ampere
export VLLM_ATTENTION_BACKEND=TRITON_ATTN_VLLM_V1
export TORCH_CUDA_ARCH_LIST="8.0"

# Launch Server
python3 -m vllm.entrypoints.openai.api_server \
  --model openai/gpt-oss-120b \
  --trust-remote-code \
  --host 0.0.0.0 \
  --port 8000 \
  \
  # Memory & Quantization Settings
  --gpu-memory-utilization 0.92 \
  --max-model-len 128000 \
  --kv-cache-dtype fp8 \
  \
  # Performance Tuning
  --max-num-batched-tokens 4096 \
  --max-num-seqs 256 \
  --async-scheduling \
```

```
\
# Stability Safeguards
--no-enable-prefix-caching \
--enforce-eager \
\
# Parallelism (Single Card)
--tensor-parallel-size 1
```

Configuration Notes:

- **--gpu-memory-utilization 0.92**: Increases the memory allocation fraction from the default 0.90, reclaiming ~1.6 GB of VRAM.
- **--enforce-eager**: Disables CUDA Graph capture. While CUDA Graphs improve performance, they can be a source of instability with the Marlin kernel on Ampere in older vLLM versions. If stability is solid, this can be removed to gain small CPU-overhead reductions.
- **--trust-remote-code**: Required for gpt-oss as it uses custom modeling code.

---

9. Economic Landscape: A100 vs. H100 TCO Analysis

The final component of the research goal is a cost-performance analysis. Is it economically rational to remain on the A100, or does the H100's performance justify its premium?

9.1 Throughput vs. Cost Matrix

The following table compares the theoretical performance and spot market costs (projected 2025/2026 rates) for the two hardware platforms.

Metric	NVIDIA A100-80GB (Optimized)	NVIDIA H100-80GB (Optimized)	Delta
System Throughput	~5,000 TPS	~10,000 TPS	H100 is 2x faster
Latency (TPOT)	~100 ms	~50-60 ms	H100 is ~2x lower latency
Spot Price / Hour	$1.75 - $2.20	$2.85 - $4.00+	H100 is ~1.75x more expensive

Cost per 1M Tokens	**~$0.11**	**~$0.097**	**H100 is ~12% cheaper**

9.2 The "Utilization Trap"

The data suggests that the H100 delivers a lower cost-per-token (~$0.097 vs $0.11), roughly a 12% saving. However, this calculation assumes **100% hardware utilization**.

- To achieve the H100's efficiency, the system must generate ~36 million tokens per hour (10,000 TPS).
- If the workload is sporadic or moderate (e.g., 2,000 TPS), the H100 sits idle half the time but costs nearly double per hour.
- **The A100 Advantage:** For workloads that do not require hyperscale throughput, the A100 is "right-sized." It costs significantly less per hour ($1.75 vs $3.00+) to keep the service available.

9.3 Market Volatility

The spot market for H100s is highly volatile. Reports indicate "price spikes" of 10% or more in short periods due to supply constraints.[19] The A100 market is far more mature and stable, with prices in North America clustering tightly around $1.75–$1.93/hr.[7]

Conclusion:

- **Stay on A100 if:** Your aggregate demand is < 15M tokens/day, or if you are cost-sensitive and cannot guarantee full saturation of an H100.
- **Migrate to H100 if:** Latency is critical (need <60ms TPOT) or if demand exceeds the capacity of a single A100 and you are considering a multi-GPU A100 cluster. A single H100 is generally better than 2x A100s due to simpler management and lower interconnect overhead.

10. Conclusion

The successful deployment of gpt-oss-120b on a single NVIDIA A100-80GB is a testament to the efficacy of the MXFP4 quantization standard and the ingenuity of the Marlin kernel. By emulating 4-bit execution in shared memory, vLLM allows previous-generation hardware to punch significantly above its weight class.

However, this capability is fragile. It relies on a precise alignment of vLLM version (v0.10.2), attention backend (TRITON_ATTN), and configuration settings. The severe instability of Prefix Caching in this version poses a significant operational risk that must be mitigated by disabling the feature.

Ultimately, the A100-80GB remains a highly attractive platform for gpt-oss-120b. It offers a

compelling balance of memory capacity, computational throughput, and market availability, providing a cost-effective alternative to the H100 for all but the most latency-critical or hyperscale applications.

Works cited

1. GPT-OSS 120B Throughput Tokens Per Second H100 - BytePlus, accessed January 26, 2026, https://www.byteplus.com/en/topic/577615
2. The most economical ways to run gpt-oss-120B, accessed January 26, 2026, https://developer.tenten.co/the-most-economical-ways-to-run-gptoss120b
3. MARLIN: Mixed-Precision Auto-Regressive Parallel Inference on Large Language Models - arXiv, accessed January 26, 2026, https://arxiv.org/html/2408.11743v1
4. Optimizing GPT-OSS-120B Throughput on NVIDIA A100 ... - GPUStack, accessed January 26, 2026, https://docs.gpustack.ai/2.0/performance-lab/gpt-oss-120b/a100/
5. [Bug]: Prefix caching with larger batch-sizes and large prompts is ..., accessed January 26, 2026, https://github.com/vllm-project/vllm/issues/22808
6. [Bug]: Prefix Cache Corruption with LoRA with the same name but different id · Issue #30931 · vllm-project/vllm - GitHub, accessed January 26, 2026, https://github.com/vllm-project/vllm/issues/30931
7. The Geography of GPU Pricing: What A100 vs H100 Tells Us About the Global Compute Market - Silicon Data, accessed January 26, 2026, https://www.silicondata.com/blog/geography-of-gpu-pricing-a100-vs-h100
8. NVIDIA H100 Price Guide 2026: GPU Costs, Cloud Pricing & Buy vs Rent, accessed January 26, 2026, https://docs.jarvislabs.ai/blog/h100-price
9. AI GPU Rental Market Trends December 2025: Complete Industry Analysis, accessed January 26, 2026, https://www.thundercompute.com/blog/ai-gpu-rental-market-trends
10. MXFP4 and various hardware : r/LocalLLaMA - Reddit, accessed January 26, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1mij7ki/mxfp4_and_various_hardware/
11. FP8 W8A8 - vLLM, accessed January 26, 2026, https://docs.vllm.ai/en/stable/features/quantization/fp8/
12. Compressing Large Neural Networks - ISTA Research Explorer, accessed January 26, 2026, https://research-explorer.ista.ac.at/download/17485/17880/frantar_thesis_final.pdf
13. MARLIN: Mixed-Precision Auto-Regressive Parallel Inference on Large Language Models - arXiv, accessed January 26, 2026, https://arxiv.org/pdf/2408.11743
14. How Marlin pushes the boundaries of mixed-precision LLM inference | Red Hat Developer, accessed January 26, 2026, https://developers.redhat.com/articles/2024/04/17/how-marlin-pushes-boundaries-mixed-precision-llm-inference
15. openai/gpt-oss-120b · vLLM FlashAttention3 with A6000 - Hugging Face, accessed January 26, 2026, https://huggingface.co/openai/gpt-oss-120b/discussions/33

16. GPT OSS - vLLM Recipes, accessed January 26, 2026, https://docs.vllm.ai/projects/recipes/en/latest/OpenAI/GPT-OSS.html
17. Which ATTENTION BACKEND for gpt-oss in version 0.11.0? - Model Support - vLLM Forums, accessed January 26, 2026, https://discuss.vllm.ai/t/which-attention-backend-for-gpt-oss-in-version-0-11-0/1691
18. Is --enable-expert-parallel supported for gpt-oss models on b200/gb200? - vLLM Forums, accessed January 26, 2026, https://discuss.vllm.ai/t/is-enable-expert-parallel-supported-for-gpt-oss-models-on-b200-gb200/1480
19. H100 Price Spike: Understanding the 10% Surge in GPU Rental Costs - Silicon Data, accessed January 26, 2026, https://www.silicondata.com/blog/h100-price-spike