Technical Evaluation: Migration Strategy for gpt-oss-120b Inference on High-Performance Infrastructure

1. Executive Summary and Strategic Imperative

This comprehensive technical report evaluates the viability and implementation strategy for migrating the inference stack of the openai/gpt-oss-120b model from vLLM to llama.cpp within a high-performance computing environment utilizing NVIDIA H100 and H200 accelerators. The primary driver for this migration is a critical stability failure in the current vLLM deployment, specifically regarding the parsing of the openai-harmony response format during long-context agentic interactions.

The analysis confirms that llama.cpp provides a technically superior runtime environment for this specific use case, offering deterministic output control via Grammar-Based Normalization Form (GBNF) that effectively resolves the parser drift issues observed in vLLM. Furthermore, llama.cpp offers unique memory management capabilities—specifically native support for MXFP4 quantization, low-bit KV cache quantization, and fine-grained expert offloading—that render it possible to serve the full 128,000-token context window on a single 80GB H100 GPU, a feat that is architecturally constrained on standard dense inference engines.

However, this migration introduces distinct engineering challenges. The report identifies a critical "gibberish" instability bug affecting H100 architectures when legacy compilation flags are used, requiring a custom build process. It also highlights the trade-off between the batched throughput of vLLM and the single-stream latency consistency of llama.cpp. The following sections detail the architectural deconstruction, performance benchmarks, and a rigorous implementation guide to execute this transition with zero downtime and maximum reliability.

---

2. Architectural Deconstruction: Model, Format, and Hardware

To understand the necessity of this migration, one must first deconstruct the interacting components: the non-standard architecture of the gpt-oss-120b model, the strict requirements of the Harmony protocol, and the physical constraints of the NVIDIA Hopper architecture.

2.1 The gpt-oss-120b Mixture-of-Experts Architecture

The gpt-oss-120b model represents a significant departure from standard dense Large Language Models (LLMs) like Llama 3 or Mistral. It is built upon a Mixture-of-Experts (MoE) architecture designed to decouple total parameter count from active computational cost.[1]

2.1.1 Parameter Sparsity and Routing

The model boasts a total parameter count of approximately **116.8 billion**. In a dense model, every parameter would participate in every forward pass, requiring massive compute resources per token. However, gpt-oss-120b utilizes a fine-grained routing mechanism with **128 experts**.[2]

- **Active Parameters:** For any given token, the router selects only the top 4 experts. This results in only **5.1 billion active parameters** per forward pass.[1]
- **Implication for Inference:** The inference characteristics resemble a 5B–7B parameter model in terms of FLOPs (Floating Point Operations), but the memory characteristics remain that of a 120B model. The inference engine must have access to the full 117B parameter set in VRAM (or high-speed system RAM) to fetch the relevant experts on demand. This makes the workload heavily **memory-bandwidth bound** rather than compute-bound.

2.1.2 Native MXFP4 Quantization

A defining feature of gpt-oss-120b is its native quantization format, **MXFP4 (Micro-exponent Floating Point 4-bit)**.[4] Unlike post-training quantization (PTQ) where a model trained in FP16 is compressed later, often incurring precision loss, gpt-oss-120b was trained with this format in mind.

- **Mechanism:** MXFP4 shares exponents across blocks of weights (block quantization), allowing the weights themselves to be stored in 4-bit integers while maintaining a dynamic range closer to FP16.
- **Storage Efficiency:** This compression reduces the model's VRAM footprint from ~240 GB (at FP16) to approximately **63 GB**.[1] This critical reduction allows the model weights to fit entirely within the 80GB memory envelope of a single NVIDIA H100 or H200 GPU, provided the inference engine can natively decode this format without upcasting the entire model to FP16 in memory.

2.2 The Harmony Response Protocol

The user's reported instability stems largely from the "Harmony" response format. This is an XML-like schema developed by OpenAI to structure agentic interactions, separating internal reasoning, tool calls, and user-facing outputs into distinct "channels".[7]

Structure of a Harmony Response:

```
XML


<|start|>assistant
<|channel|>analysis<|message|>

<|end|>
<|channel|>commentary<|message|>
  to=functions.tool_name
  <|constrain|>json<|message|>{"arg": "value"}
<|end|>
<|channel|>final<|message|>

<|end|>
```

The Parsing Challenge:

In a standard inference pipeline (like vLLM), the model generates these tokens probabilistically. As the context window fills (approaching 100k+ tokens), the attention mechanism's precision can degrade slightly (attention sink drift). The model might hallucinate a variation of a tag—for example, generating <|channel > (missing a pipe) or <channel> (missing pipes entirely).

- **Fragility:** The openai-harmony parser [7] expects rigid adherence to the schema. A single malformed tag can cause the parser to fail to identify a tool call, treating it as plain text or throwing an exception.
- **Consequence:** In autonomous agent loops, this parsing failure breaks the execution chain. The agent "forgets" to call the tool or the system fails to parse the tool arguments, leading to the conversation stalling or spiraling into repetition.

2.3 Hardware Constraints: NVIDIA H100/H200

The deployment targets NVIDIA's Hopper architecture (H100/H200). While these are the most powerful accelerators currently available, they have specific characteristics that influence the choice of inference engine.

- **Memory Bandwidth:** The H100 SXM5 offers 3.35 TB/s of bandwidth. This is sufficient to feed the active experts of the gpt-oss-120b model, provided the data is resident in HBM3 memory.
- **Compute Capability 9.0:** The H100 introduces the Transformer Engine with native FP8 support. However, native **FP4** support (which would map directly to MXFP4) is a feature of the next-generation **Blackwell (B200, sm_100/sm_120)** architecture.[8]
- **Emulation Overhead:** On the H100, MXFP4 weights must be dequantized on-the-fly to

BF16 or FP16 before computation in the Tensor Cores. This dequantization step requires highly optimized CUDA kernels to avoid becoming a bottleneck. llama.cpp includes custom MMQ (Matrix-Matrix Quantized) kernels specifically optimized for this [9], whereas vLLM often relies on generic library calls that may not be as tightly optimized for this specific niche format on Hopper hardware.

---

3. Comparative Analysis: vLLM vs. llama.cpp

The decision to migrate rests on the fundamental architectural differences between vLLM and llama.cpp. While vLLM is designed for high-throughput batch serving, llama.cpp prioritizes flexibility, edge compatibility, and latency control.

3.1 Throughput vs. Latency Philosophy

vLLM (High Throughput, High Jitter):

vLLM utilizes PagedAttention to manage memory fragmentation, allowing it to batch hundreds of concurrent requests. It prioritizes aggregate throughput (tokens per second across all users) over the latency of any single request.

- **Queueing:** To maximize GPU utilization, vLLM may delay the processing of a single user's token generation to batch it with others. In an agentic workflow involving a single, long-running conversation, this can introduce variable latency (jitter).[11]
- **Resource Greed:** vLLM aggressively reserves VRAM for the KV cache to support these large batches. This leaves less room for the model weights themselves, making it difficult to fit the 128k context of gpt-oss-120b on a single GPU without aggressive tensor parallelism (TP) across multiple cards.

llama.cpp (Consistent Latency, Single-Stream Focus):

llama.cpp executes a static compute graph. It is optimized for running a single model instance efficiently, often with a batch size of 1 (single user).

- **Predictability:** For an agentic workload where the "user" is an automated script waiting for a tool call, llama.cpp provides consistent time-to-first-token (TTFT) and inter-token latency.[11]
- **Efficiency:** It uses a custom memory allocator that is less aggressive than vLLM's, allowing for tighter packing of the model and context into VRAM.

3.2 The Stability Divergence: Parsing vs. Sampling

The most critical differentiator for this research task is how each engine handles the Harmony format.

- **vLLM (Post-Generation Parsing):** vLLM generates tokens based purely on probability. It is the responsibility of the client (or an intermediate layer) to parse the output. If the

model generates a syntax error, the generation has already happened, and the error is fatal to the parser.

- **llama.cpp (Pre-Generation Constraint):** llama.cpp supports **GBNF (Grammar-Based Normalization Form)**. This allows the user to define the Harmony schema as a formal grammar.
  - **Mechanism:** During the sampling phase (before a token is selected), the engine checks the grammar state. Any token that would violate the grammar is masked out (assigned a probability of negative infinity).
  - **Implication:** The model is *forced* to generate valid Harmony tags. It effectively "fixes" the model's drift by constraining its output space to only valid syntactic constructs.[12] This resolves the "parser failure" issue at the source.

---

4. Stability Engineering: The Harmony Grammar Solution

To address the parser failures reported in the user's current setup, the migration to llama.cpp must utilize its grammar capabilities. Relying on prompt templates alone is insufficient for long-context stability.

4.1 Deconstructing the Failure Mode

In deep context (100k+ tokens), the attention mechanism's "needle in a haystack" retrieval capability remains robust for semantics but can degrade for rigid syntax. The model might recall that it needs to output a tool call but might "forget" the exact number of pipe characters in <|channel|>.

- **The vLLM Failure:** The openai-harmony parser uses regular expressions or strict string matching. A missing character breaks the match, causing the tool call to be missed. The agent, expecting a tool output, might re-prompt the model, leading to a loop where the model repeatedly outputs the invalid syntax.[14]

4.2 Implementing GBNF for Harmony

The solution is to provide llama.cpp with a grammar file that strictly defines the Harmony syntax.

Grammar Logic:

1. **Root:** The response must consist of an optional analysis block, followed by a tool call (commentary) OR a final response.
2. **Analysis Channel:** Must start with <|channel|>analysis<|message|>, contain text, and end with <|end|>.
3. **Commentary Channel:** Must start with <|channel|>commentary<|message|>, followed by

to=functions.name, and then a constrained JSON object.

Recommended harmony.gbnf Content [12]:

Code snippet

```
root      ::= (analysis)? (tool_call | final_response)
analysis  ::= "<|channel|>analysis<|message|>" content "<|end|>"
content   ::= [^<]+

tool_call ::= "<|start|>assistant" "<|channel|>commentary<|message|>" "to=functions."
[a-zA-Z0-9_]+ ws "<|constrain|>json<|message|>" json_object "<|end|>"

final_response ::= "<|start|>assistant" "<|channel|>final<|message|>" content "<|end|>"

json_object ::= "{" ws (json_member ("," ws json_member)*)? "}"
json_member ::= string ":" ws value
string      ::= "\"" [^"]* "\""
value       ::= string | number | json_object | array | "true" | "false" | "null"
ws          ::= [ \t\n]*
```

Note: This is a simplified representation. The actual GBNF implementation must handle all edge cases of the Harmony format.

By launching llama-server with --grammar-file harmony.gbnf, the engine enforces this structure. If the model predicts a probability distribution where "invalid syntax" is the most likely token, the sampler suppresses it and selects the next most likely *valid* token. This ensures 100% syntactic compliance, stabilizing the agentic loop even at 128k context.

---

5. Memory Engineering: Achieving 128k Context on H100

The most significant technical hurdle is fitting the 128k context window on an 80GB GPU alongside the 117B parameter model. Standard FP16 calculations suggest this is impossible. We must employ advanced memory management techniques available in llama.cpp.

5.1 The Memory Math of Impossibility

Let us rigorously calculate the memory requirements for a standard deployment.

Model Weights:

- **Format:** MXFP4 (~4.25 bits effective per parameter).
- **Size:** 116.8 Billion parameters.
- **Weight Memory:** ~63 GB.[1]
- *Remaining VRAM on H100 (80GB):* ~17 GB.

KV Cache (FP16):

The KV cache stores the key and value matrices for every token in the context to avoid recomputation.

- **Formula:** 2 * n_layers * n_kv_heads * head_dim * n_tokens * bytes_per_element.
- **Parameters:**
  - n_layers: 36
  - n_kv_heads: 8 (Due to Grouped Query Attention [15])
  - head_dim: 128
  - bytes_per_element: 2 (FP16)
- **Calculation:**
  - 2 * 36 * 8 * 128 * 131,072 * 2 = 19,327,352,832 bytes
  - **Total KV Cache Size: ~19.3 GB**.

The Deficit:

- Total Required: 63 GB (Weights) + 19.3 GB (KV) + ~2 GB (Activation Overhead) = **~84.3 GB**.
- Total Available: 80 GB.
- **Deficit:** ~4.3 GB.

Direct deployment on a single H100 is mathematically impossible with full precision KV cache.

5.2 Solution Strategy A: KV Cache Quantization

llama.cpp supports storing the KV cache in quantized formats (Q8_0 or Q4_0) rather than FP16. This is a highly effective optimization for long-context models.

- **Q8_0 (8-bit):** Reduces KV cache size by 50%.
  - New KV Size: ~9.65 GB.
  - Total VRAM: 63 + 9.65 + 2 = **~74.65 GB**.
  - *Feasibility:* Fits comfortably on an 80GB H100.
- **Q4_0 (4-bit):** Reduces KV cache size by 75%.
  - New KV Size: ~4.8 GB.
  - Total VRAM: 63 + 4.8 + 2 = **~69.8 GB**.
  - *Feasibility:* Fits with significant headroom for batching.

Impact on Quality: Research suggests that Q8_0 KV cache has negligible impact on perplexity or reasoning capabilities.[16] Q4_0 shows slight degradation in "needle in a haystack" retrieval tasks but is generally robust for conversational context.

- **Recommendation:** Use **Q8_0** (--cache-type-k q8_0 --cache-type-v q8_0) as the primary strategy to ensure maximum reasoning fidelity while fitting within the 80GB envelope. If VRAM pressure increases (e.g., larger batch sizes), drop to Q4_0.

5.3 Solution Strategy B: MoE Expert Offloading

If strict FP16 KV cache precision is required, llama.cpp offers a unique optimization for MoE models: **Expert Offloading**.

- **Architecture:** In an MoE model, the majority of parameters (the experts) are idle for any given token. Only the "Attention" layers and the "Router" are constantly active.
- **Mechanism:** llama.cpp can keep the Attention layers on the GPU (fast access) while storing the Expert weights in system RAM (CPU). When a token requires specific experts, they are fetched over the PCIe bus.
- **PCIe Bottleneck:** The H100 connects via PCIe Gen5 x16 (~63 GB/s bandwidth).
  - If we offload 20% of the experts (~12GB) to CPU RAM, we free up 12GB of VRAM.
  - However, fetching these weights for every token generates traffic. If the active experts are in RAM, generation slows down.
  - *Performance Impact:* Moving from full GPU to partial offload typically drops generation speed from ~30 t/s to ~10–15 t/s.[17]

Verdict: KV Cache Quantization (Strategy A) is superior to Expert Offloading (Strategy B) because it maintains the model weights in high-bandwidth HBM3 memory, preserving the high token generation speed of the H100. Expert offloading should be a fallback only if quantization proves unacceptable for accuracy.

---

6. Critical Bug Analysis: The "Gibberish" Instability

A comprehensive review of the research material uncovers a critical instability affecting gpt-oss-120b on H100 and RTX 4090/5090 architectures. This is a "showstopper" bug that must be proactively mitigated during the build process.

6.1 Symptoms

Users report that after generating a certain number of tokens (often in long contexts >7k), the model output devolves into repeating characters (e.g., "GGGGGG...") or nonsensical garbage.[18] This behavior is distinct from the parser loops described earlier; it is a fundamental numerical collapse in the inference engine.

6.2 Root Cause Analysis

The issue has been traced to the interaction between the GGML_CUDA_FORCE_CUBLAS flag and the MXFP4 dequantization kernels on Compute Capability 8.9+ (Ada/Hopper/Blackwell) architectures.[10]

- **Legacy Behavior:** Historically, llama.cpp relied on cuBLAS (NVIDIA's Basic Linear Algebra Subprograms) for maximum performance on dense matrices.
- **The Conflict:** The MXFP4 format requires specific custom kernels (MMQ - Matrix-Matrix Quantized) implemented within llama.cpp to handle the 4-bit block dequantization correctly. When GGML_CUDA_FORCE_CUBLAS is enabled, the engine attempts to force these operations through standard cuBLAS paths that may expect standard layouts or precisions (FP16/BF16), leading to numerical overflow or misalignment in the tensor cores.
- **Architecture Specifics:** This manifests primarily on newer cards (H100/RTX4090) because their Tensor Cores handle FP16/BF16 accumulation differently than older Pascal/Volta cards, making the precision mismatch fatal.

6.3 Mitigation Strategy

The fix is strictly compile-time. The deployment MUST build llama.cpp from source with specific flags to disable the legacy cuBLAS forcing and rely on the modern, architecture-aware MMQ kernels.

Required CMake Flags:

- -DGGML_CUDA_FORCE_CUBLAS=OFF: Disables the conflicting library path.
- -DGGML_CUDA_F16=OFF: Forces FP32 accumulation in certain operations to prevent overflow, a key factor in the "gibberish" bug for long contexts.[18]

---

7. Implementation Specification: Docker & Runtime

To execute the migration, a custom Docker environment is required. Standard "latest" images from Docker Hub are insufficient because they may not have the H100-specific flags or the critical "gibberish" fix enabled by default.

7.1 Optimized Dockerfile for H100

This Dockerfile compiles llama.cpp specifically for the H100 (sm_90) architecture, ensuring native support for MXFP4 and applying the stability fixes.

Dockerfile

```
# Use the NVIDIA CUDA 12.4 development base image.
# H100/Hopper requires CUDA 12.x for optimal performance.
FROM nvidia/cuda:12.4.1-devel-ubuntu22.04

# Install essential build dependencies and curl for API checks.
RUN apt-get update && apt-get install -y \
    git build-essential cmake python3 python3-pip \
    libcurl4-openssl-dev curl \
    && rm -rf /var/lib/apt/lists/*

# Set working directory
WORKDIR /app

# Clone llama.cpp.
# We target a specific commit or master to ensure we have the MXFP4 fixes.
# As of Q1 2026, master is generally stable, but pinning a hash is recommended for prod.
RUN git clone https://github.com/ggml-org/llama.cpp.git. \
    && git checkout master

# COMPILATION STAGE
# Critical Flags Explanation:
# -DGGML_CUDA=ON: Enable NVIDIA GPU support.
# -DCMAKE_CUDA_ARCHITECTURES=90: Target H100 (Hopper). Do NOT use 120 (Blackwell).
# -DGGML_CUDA_FORCE_CUBLAS=OFF: FIXES THE GIBBERISH BUG.
# -DGGML_CUDA_F16=OFF: Prevents FP16 overflow in long contexts.
# -DLLAMA_CURL=ON: Enables downloading models directly from HF (optional but useful).
RUN cmake -B build \
    -DGGML_CUDA=ON \
    -DCMAKE_CUDA_ARCHITECTURES=90 \
    -DGGML_CUDA_FORCE_CUBLAS=OFF \
    -DGGML_CUDA_F16=OFF \
    -DLLAMA_CURL=ON \
    -DCMAKE_BUILD_TYPE=Release \
    && cmake --build build --config Release -j $(nproc)

# Expose the server port
EXPOSE 8080

# Define the entrypoint
ENTRYPOINT ["./build/bin/llama-server"]
```

7.2 Runtime Configuration

The startup command governs the memory management strategies discussed in Section 5.

Command Structure:

Bash

```
./llama-server \
 --model /models/gpt-oss-120b-MXFP4-00001-of-00002.gguf \
 --host 0.0.0.0 --port 8080 \
 --ctx-size 131072 \
 --n-gpu-layers 999 \
 --batch-size 2048 \
 --ubatch-size 512 \
 --flash-attn \
 --cache-type-k q8_0 \
 --cache-type-v q8_0 \
 --grammar-file /app/harmony.gbnf \
 --jinja \
 --chat-template-kwargs '{"reasoning_effort": "high"}'
```

Parameter Breakdown:

Flag	Value	Purpose
--model	...MXFP4...gguf	Path to the natively quantized model file.
--ctx-size	131072	Sets the context window to the full 128k capability.
--n-gpu-layers	999	Instructions to offload all *model layers* to the GPU.
--flash-attn	(Flag)	Enables Flash Attention kernels. **Mandatory** for 128k context performance.
--cache-type-k/v	q8_0	Quantizes KV cache to 8-bit to fit in 80GB VRAM

		(See Section 5.2).
--grammar-file	harmony.gbnf	Enforces the Harmony syntax to prevent parser failures (See Section 4).
--chat-template-kwargs	{"reasoning_effort": "high"}	Configures the model to use maximum reasoning depth.[19]

7.3 Operational Benchmarks (H100)

Based on the integration of native MXFP4 support in llama.cpp (PR #17906), the following performance metrics are expected on an H100 80GB:

- **Prompt Processing (Prefill):** ~500–800 tokens/second. (Slower than vLLM, but optimized via Flash Attention).
- **Token Generation (Decode): 30–45 tokens/second**.[20]
  - *Note:* This is significantly faster than human reading speed and sufficient for rapid tool-use interactions.
- **Latency Stability:** High consistency. Unlike vLLM, which might pause generation to batch other requests, llama.cpp will deliver a steady stream of tokens, crucial for the "feeling" of responsiveness in an interactive agent.

---

8. Conclusion and Strategic Recommendation

The investigation into replacing vLLM with llama.cpp for the gpt-oss-120b workload yields a definitive positive recommendation, provided specific engineering protocols are followed.

Summary of Findings:

1. **Stability:** llama.cpp with GBNF grammars offers a deterministic solution to the parser failure problem, superior to vLLM's probabilistic approach.
2. **Feasibility:** The 128k context window is deployable on a single H100 only via llama.cpp's support for KV cache quantization (Q8_0/Q4_0), resolving the memory deficit that plagues dense FP16 engines.
3. **Performance:** While aggregate throughput is lower than vLLM, the single-stream performance (30+ t/s) is more than adequate for the defined use case.

Deployment Roadmap:

1. **Immediate Action:** Build the custom Docker image defined in Section 7.1. Do not use pre-built binaries.
2. **Validation:** Deploy the container with the harmony.gbnf grammar and run a regression

test on the long-context agentic prompts that previously caused failures.
3. **Tuning:** Monitor VRAM usage. If the H100 saturates, downgrade KV cache to q4_0. If "gibberish" output appears, verify the GGML_CUDA_FORCE_CUBLAS=OFF flag was applied.

By adopting this strategy, the infrastructure will transition from a fragile, high-throughput setup to a robust, consistent, and architecturally optimized environment capable of maximizing the unique potential of the gpt-oss-120b MoE model.

Works cited

1. gpt-oss-120b (free) - API, Providers, Stats - OpenRouter, accessed January 25, 2026, https://openrouter.ai/openai/gpt-oss-120b:free
2. OpenAI GPT-OSS 120B - GroqDocs - Groq Console, accessed January 25, 2026, https://console.groq.com/docs/model/openai/gpt-oss-120b
3. config.json · openai/gpt-oss-120b at main - Hugging Face, accessed January 25, 2026, https://huggingface.co/openai/gpt-oss-120b/blob/main/config.json
4. Introducing gpt-oss - OpenAI, accessed January 25, 2026, https://openai.com/index/introducing-gpt-oss/
5. MXFP4, FP4, and FP8: How GPT-OSS Runs 120B Parameters on an 80GB GPU with MoE Weight Quantization - Abdullah Grewal, accessed January 25, 2026, https://buzzgrewal.medium.com/mxfp4-fp4-and-fp8-how-gpt-oss-runs-120b-parameters-on-an-80gb-gpu-with-moe-weight-quantization-db26b57fd787
6. Why are the quants for gpt-oss-120b all roughly the same size? : r/LocalLLaMA - Reddit, accessed January 25, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1mr0giq/why_are_the_quants_for_gptoss120b_all_roughly_the/
7. OpenAI Harmony Response Format, accessed January 25, 2026, https://developers.openai.com/cookbook/articles/openai-harmony
8. Does blackwell/new GPU matter to train model with MXFP4 ? : r/LocalLLaMA - Reddit, accessed January 25, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1oo5z3w/does_blackwellnew_gpu_matter_to_train_model_with/
9. llama.cpp, experimental native mxfp4 support for blackwell (25% preprocessing speedup!) : r/LocalLLaMA - Reddit, accessed January 25, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1pwybg6/llamacpp_experimental_native_mxfp4_support_for/
10. Running gpt-oss-120b model with llama.cpp on H100 GPUs? #16198 - GitHub, accessed January 25, 2026, https://github.com/ggml-org/llama.cpp/discussions/16198
11. vLLM or llama.cpp: Choosing the right LLM inference engine for your use case, accessed January 25, 2026, https://developers.redhat.com/articles/2025/09/30/vllm-or-llamacpp-choosing-right-llm-inference-engine-your-use-case
12. Making GPT-OSS 20B and CLine work together. - Reddit, accessed January 25,

2026, https://www.reddit.com/r/CLine/comments/1mtcj2v/making_gptoss_20b_and_cline_work_together/
13. gpt-oss and grammar #15341 - ggml-org llama.cpp - GitHub, accessed January 25, 2026, https://github.com/ggml-org/llama.cpp/discussions/15341
14. VLLM v. Llama.cpp for Long Context on RTX 5090 : r/LocalLLaMA - Reddit, accessed January 25, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1nnv17u/vllm_v_llamacpp_for_long_context_on_rtx_5090/
15. Things to Know About OpenAI GPT-OSS: Run it Locally on Your Device, Hardware Requirements, Performance Guide, and Model Architecture Explained | by Isaak Kamau | Medium, accessed January 25, 2026, https://medium.com/@isaakmwangi2018/things-to-know-about-openai-gpt-oss-run-it-locally-on-your-device-hardware-requirements-e266e0f1700f
16. What's the verdict on using quantized KV cache? : r/LocalLLaMA - Reddit, accessed January 25, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1mhlj69/whats_the_verdict_on_using_quantized_kv_cache/
17. How to properly run gpt-oss-120b on multiple GPUs with llama.cpp? : r/LocalLLaMA - Reddit, accessed January 25, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1pjv5wz/how_to_properly_run_gptoss120b_on_multiple_gpus/
18. Eval bug: Gibberish with long context prompts GPT OSS · Issue ..., accessed January 25, 2026, https://github.com/ggml-org/llama.cpp/issues/15112
19. Is there any wayto change reasoning effort on the fly for GPT-OSS in llama.cpp? - Reddit, accessed January 25, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1o9584a/is_there_any_wayto_change_reasoning_effort_on_the/
20. How's your experience with the GPT OSS models? Which tasks do you find them good at—writing, coding, or something else : r/LocalLLaMA - Reddit, accessed January 25, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1n3u7qf/hows_your_experience_with_the_gpt_oss_models/
21. llama.cpp recent updates - gpt120 = 20t/s : r/LocalLLaMA - Reddit, accessed January 25, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1po6we5/llamacpp_recent_updates_gpt120_20ts/
22. Llama.cpp experimental native mxfp4 support for blackwell PR - DGX Spark / GB10, accessed January 25, 2026, https://forums.developer.nvidia.com/t/llama-cpp-experimental-native-mxfp4-support-for-blackwell-pr/355639