Optimization of GPT-OSS-120B Inference in Llama.cpp: Resolving Sliding Window Attention and KV Cache Reuse Conflicts

Executive Summary

The deployment of large-scale open-weights models, specifically the gpt-oss-120b variant, within local inference environments such as llama.cpp represents a significant milestone in democratized AI. However, this deployment is currently hindered by a critical incompatibility between the model's architectural specifications—namely Sliding Window Attention (SWA)—and the inference engine's memory management logic for Key-Value (KV) cache reuse. This report provides an exhaustive technical analysis of the issue where llama.cpp disables cache reuse for gpt-oss-120b, resulting in severe performance degradation during multi-turn interactions.

The investigation confirms that the root cause is a deterministic safeguard within the llama.cpp runtime. The engine identifies the gpt-oss.attention.sliding_window parameter (set to 128 tokens) in the GGUF metadata and explicitly disables prefix caching to prevent potential context corruption, as evidenced by the log message: *"cache_reuse is not supported by this context, it will be disabled"*.[1] This behavior forces a full reprocessing of the prompt for every request, rendering the system inefficient for agentic workflows or long-context chat.

Through detailed architectural dissection, log analysis, and review of engineering workarounds, this report identifies the primary solution: the modification of the GGUF model file to remove or nullify the sliding window metadata. This intervention forces the inference engine to adopt a full-attention memory model, thereby enabling the standard prefix caching algorithms to function. Secondary solutions involving runtime flags (--swa-full) and hardware-specific offloading strategies are also evaluated. The report concludes with a comprehensive guide to implementing these fixes, optimizing runtime parameters, and configuring hardware to maximize the throughput of this 120-billion parameter Mixture-of-Experts (MoE) model.

---

1. Introduction and Problem Context

1.1 The Operational Landscape of Large-Scale Local Inference

The landscape of local Large Language Model (LLM) inference has evolved rapidly, moving from experimental deployments of 7B parameter models to the hosting of massive, datacenter-class models like gpt-oss-120b on consumer and workstation hardware. This shift is enabled by two primary technologies: aggressive quantization (e.g., MXFP4, Q4_K) and efficient inference engines like llama.cpp that support heterogeneous compute (CPU/GPU splitting).[3]

However, as models grow in complexity, they incorporate architectural features designed to mitigate the computational cost of their scale. gpt-oss-120b utilizes a Mixture-of-Experts (MoE) architecture to reduce active parameter count and Sliding Window Attention (SWA) to cap the memory complexity of attention operations.[5] While these features are beneficial for the model's native runtime environment, they present integration challenges for general-purpose inference engines that must abstract these complexities into a unified memory model.

1.2 The Specific Problem: Latency in Multi-Turn Workflows

The core utility of an LLM in a chat or agentic context relies on its ability to "remember" the conversation history without re-reading it from scratch every time. This is technically achieved through **KV Cache Reuse** (also known as prefix caching). When a user sends a new message in a conversation, the prompt consists of . An optimized engine recognizes that has already been processed and resides in the GPU memory (VRAM). It reuses these stored Key and Value tensors, processing only the [New Message].

For gpt-oss-120b, users report a catastrophic failure of this mechanism. Despite enabling --cache-reuse, the engine explicitly rejects the optimization.[1]

- **Symptom:** Time-to-First-Token (TTFT) increases linearly with conversation length.
- **Observation:** Server logs show the n_past token count resetting to 0 for every request, indicating a complete cache flush.
- **Impact:** A conversation with 4,000 tokens of history, which should generate a response instantly, takes minutes to process on a 120B model, effectively breaking the user experience.

1.3 Research Objectives

This report addresses the following technical objectives derived from the user's query:

1. **Root Cause Determination:** definitively explain why llama.cpp disables cache reuse for this specific model architecture. Is it the MoE structure, the YaRN scaling, or the Sliding Window Attention?
2. **Solution Engineering:** Identify and validate methods to force the engine to enable cache reuse, including flag combinations, source code modifications, or GGUF file patching.
3. **Optimization Strategy:** Define the optimal configuration (context size, quantization types, offloading) to run gpt-oss-120b efficiently once cache reuse is restored.

2. Architectural Deconstruction of GPT-OSS-120B

To understand the conflict, we must first understand the machine that is gpt-oss-120b. It is not a standard dense Transformer; it is a composite of several advanced efficiency techniques.

2.1 Mixture of Experts (MoE)

The model follows a sparse MoE architecture.

- **Total Parameters:** ~120 Billion.
- **Experts:** 128 experts per layer.[5]
- **Active Experts:** Likely top-2 routing, meaning for any given token, only a fraction of the parameters (the "active set") are loaded and computed.

Relevance to Cache Reuse: MoE models are notoriously memory-bandwidth bound. While the compute (FLOPs) required to process a token is lower than a dense 120B model, the model size requires pulling vast amounts of weight data from VRAM to the compute units. This makes the *prefill* phase (processing the prompt) particularly expensive if the weights must be streamed from system RAM (CPU offloading). Therefore, failing to reuse the cache is *more* punishing on an MoE model running on limited VRAM than on a smaller dense model, because the cost of re-reading the weights for the prompt history is exorbitant.[7]

2.2 YaRN (Yet another RoPE extensioN)

The model supports a massive context window of 131,072 tokens.[5] Standard positional embeddings decay in effectiveness at such lengths. gpt-oss employs YaRN, a method to extend the Rotary Positional Embeddings (RoPE) to handle long contexts without retraining the model on the full length.

- **Mechanism:** It modifies the frequency of the rotary embedding rotation.
- **Relevance to Cache Reuse:** YaRN is generally stateless regarding the cache structure. It affects the *values* inside the KV cache but does not dictate the *management* of the cache slots. Therefore, YaRN is unlikely to be the cause of the cache_reuse disabling.[6]

2.3 Sliding Window Attention (SWA)

This is the critical architectural feature.

- **Definition:** In SWA, a token at position t only attends to tokens in the window $$. For gpt-oss-120b, the metadata specifies gpt-oss.attention.sliding_window = 128.[5]
- **Theoretical Memory Complexity:** $O(N \times W)$ rather than $O(N^2)$.

- **GGUF Definition:** The parameter is explicitly defined in the GGUF header.

2.3.1 The "Hybrid" Nature and Attention Sinks

Research indicates that gpt-oss is not a "pure" sliding window model. It uses a hybrid approach where some layers use full attention and others use sliding window attention, or it utilizes **Attention Sinks**.

- **Attention Sinks:** The first few tokens of the sequence (the "sink") are retained in the attention window regardless of distance. This stabilizes the softmax calculation, preventing the "massive activation" issues seen when the start of the sentence falls out of view.[1]
- **Impact:** The presence of sinks implies that the model *needs* access to tokens outside the sliding window (specifically the start). However, the sliding_window metadata signals a restriction. This ambiguity is where the friction with llama.cpp begins.

---

3. Llama.cpp Inference Engine Internals

To understand why the engine rejects the model's configuration, we must look at how llama.cpp manages memory.

3.1 The KV Cache: Linear vs. Ring Buffer

The KV cache is a large block of VRAM allocated to store the Key and Value matrices for every token in the context.

- **Linear Buffer (Standard):** Memory is allocated sequentially. Token 0 goes to slot 0, Token 1 to slot 1, etc. This is simple and perfect for prefix caching. To reuse the prefix "The quick brown", the engine simply checks if slots 0, 1, and 2 contain the KV data for those tokens.
- **Ring Buffer (SWA Optimized):** For a sliding window of size W, one only needs to store W tokens. llama.cpp can implement this as a ring buffer. If $W = 128$, Token 128 overwrites Slot 0. Token 129 overwrites Slot 1.
  - *Advantage:* Massive VRAM savings. A 100k context only needs memory for 128 tokens.
  - *Disadvantage:* The physical slot index no longer corresponds to the logical token position in a static way. The "prefix" is physically destroyed as the window slides.

3.2 The Prefix Matching Algorithm

The cache_reuse feature in llama.cpp (often referred to as the Longest Common Prefix or LCP algorithm) works by comparing the token IDs of the new request against the tokens stored in the KV cache slots.[9]

1. **Input:** New tokens $$.
2. **State:** Cache contains $[C_0, C_1, C_2, ...C_m]$.
3. **Comparison:** Find index k such that $T_i == C_i$ for all $0 \leq i < k$.
4. **Action:** Retain slots $0...k-1$. Discard slots $k...end$. Append new tokens starting at k.

3.3 The Collision

The collision occurs because llama.cpp's SWA implementation and its Prefix Caching implementation operate on fundamentally different assumptions about the lifecycle of a cache slot.

If n_swa (sliding window size) is defined and non-zero:

1. The engine anticipates that it may need to evict old tokens to respect the window.
2. If the engine were to reuse the prefix, it effectively needs to store the *entire* history to ensure that the "sink" tokens and the "window" tokens are available for any arbitrary backtrack or splice.
3. However, the SWA flag signals "I am optimizing for memory, I will not store the full history."
4. **The Safeguard:** Consequently, llama.cpp contains a hard-coded check: **If Sliding Window is enabled, disable Cache Reuse.** This is done to prevent undefined behavior where the engine tries to reuse a prefix that has been partially overwritten or logically invalidated by the rolling window.[7]

The log message cache_reuse is not supported by this context is the direct output of this safeguard logic located in the llama_new_context or llama_decode initialization routines.[1]

4. Failure Analysis: The SWA/Cache Reuse Collision

4.1 Log Forensic Analysis

Let us examine the logs provided in snippet [2] to confirm this behavior.

Log Entry 1:

```
slot update_slots: id 3 | task 12914 | new prompt, n_ctx_slot = 8192, n_keep = 0,
n_prompt_tokens = 86

slot update_slots: id 3 | task 12914 | kv cache rm confirms the GGUF file contains:
```

kv 19: gpt-oss.attention.sliding_window u32 = 128`

This value of 128 is remarkably small compared to the 131,072 context length. It implies that theoretically, the model only "needs" 128 tokens of memory. However, llama.cpp users typically run these models on high-RAM servers where they *want* to use the full context. The engine is obeying the metadata ("use a tiny window") which triggers the "disable reuse" safety, despite the user's intent ("use full context and cache it").

4.3 Why Manual Flags Fail

The user attempted --cache-reuse 256 and --swa-checkpoints.

- --cache-reuse 256: This sets the threshold for reuse. It tells the engine "only reuse if 256 tokens match". However, since the feature is globally disabled by the context check, this threshold is ignored.[10]
- --swa-checkpoints: This relates to saving states for sliding windows but does not override the fundamental incompatibility in the llama_kv_cache logic.

---

5. Engineering Solutions: The Metadata Intervention

Since the root cause is the presence of the sliding_window key in the GGUF header triggering a software lockout, the most effective solution is to modify the file to remove this trigger. This forces llama.cpp to treat the model as a standard full-attention architecture.

5.1 Rationale for Metadata Modification

By removing the sliding_window key:

1. llama.cpp defaults n_swa to 0 (disabled) or n_ctx (full context).
2. The memory manager allocates a standard linear KV cache.
3. The "incompatibility check" in the initialization code returns false (no conflict).
4. Cache reuse is allowed to proceed.

Does this break the model? Theoretically, the model was trained with SWA. However, gpt-oss uses a hybrid attention scheme. Furthermore, the attention mask used during inference is permissive; allowing a token to attend to *more* history than it was trained on (Full Attention instead of Windowed) is generally benign for LLMs, especially those using RoPE/YaRN, which naturally decay attention for distant tokens. Empirical evidence from the community suggests this "lobotomy" restores performance without degrading coherence.[11]

5.2 Implementation Guide: Patching the GGUF

We utilize the gguf-new-metadata.py script provided in the llama.cpp repository.

Step 5.2.1: Tool Setup

Ensure the python scripts are available.

Bash

```
git clone https://github.com/ggml-org/llama.cpp
cd llama.cpp/gguf-py
pip install.
```

Step 5.2.2: Verification

Inspect the current metadata to confirm the key name.

Bash

```
gguf-dump.py /path/to/gpt-oss-120b.gguf | grep sliding
```

Output Expectation: gpt-oss.attention.sliding_window u32 = 128

Step 5.2.3: Execution (Removing the Metadata)

The script gguf-new-metadata.py supports removing keys. Note that this creates a copy of the file. Since the file is 59GB, ensure sufficient disk space (approx. 120GB free space required for the operation).

Bash

```
python3 gguf-new-metadata.py \
  --remove-metadata gpt-oss.attention.sliding_window \
  /path/to/gpt-oss-120b.gguf \
  /path/to/gpt-oss-120b-NO-SWA.gguf
```

Alternative In-Place Modification (Advanced):

If disk space is a constraint, advanced users might attempt to overwrite the value rather than remove the key, setting the window size to equal the context size. This can be done with gguf-set-metadata.py.

Bash

```
python3 gguf-set-metadata.py \
  /path/to/gpt-oss-120b.gguf \
  gpt-oss.attention.sliding_window \
  131072 \
  --type u32
```

Risk Warning: This modifies the file in place. If the script fails, the model file may be corrupted. Backup is essential.

5.3 Validation

After patching, launch the server with the modified file.

Bash

```
./llama-server -m /path/to/gpt-oss-120b-NO-SWA.gguf -c 131072 --cache-reuse 256
```

Monitor the logs. You should no longer see the warning cache_reuse is not supported. Subsequent requests should show n_past increasing, and kv cache rm should not trigger for the prefix.

---

6. Runtime Optimization & Configuration

Once the cache reuse mechanism is unlocked, the system must be tuned to handle the massive memory footprint of a 120B model with a 131k context.

6.1 KV Cache Quantization

This is the single most critical parameter for gpt-oss-120b.

- **The Math:** A 120B model with a 131,072 context window using FP16 (16-bit) KV cache requires:

  $$2\ (\text{Key+Value}) \times 2\ \text{bytes} \times \text{Layers} \times \text{Heads} \times \text{HeadDim} \times 131,072$$

  Snippet [15] suggests a calculation of ~37KB per token, but let's be precise. gpt-oss has 36 blocks (layers). If we use standard FP16 cache, the VRAM usage for the *context alone* could exceed 40-50GB, competing with the 59GB model weights.
- **The Solution:** Use **Q8_0** (8-bit) or **Q4_0** (4-bit) KV cache.
  - Flag: -ctk q8_0 -ctv q8_0.[1]
  - Effect: Reduces context VRAM usage by 50% (vs FP16) or 75% (vs FP32) with minimal degradation in retrieval accuracy.
  - *Note:* Snippet [1] mentioned gpt-oss disliking quantized cache due to attention sinks. If instability occurs (gibberish output after long context), revert to f16 cache but reduce context size (-c 32768). However, q8_0 is generally safe.

6.2 Flash Attention

Flash Attention is mandatory for contexts of this magnitude.

- Flag: -fa or --flash-attn.
- Benefit: Reduces the complexity of the attention calculation from quadratic memory to linear. Without this, a 131k context would likely cause an Out-Of-Memory (OOM) error during the attention score calculation phase, regardless of KV cache size.[12]

6.3 Offloading Strategies

gpt-oss-120b is an MoE model. The "weights" are huge (59GB), but the compute per token is relatively light.

- **Hardware Setup:** Assuming a setup like 2x RTX 3090 (24GB + 24GB = 48GB VRAM) or similar. The model (59GB) + Context (10GB+) = ~70GB total. This exceeds VRAM.
- **Partial Offloading:** We must offload some layers to system RAM (CPU).
- **MoE Specific Offloading:** The most efficient strategy for MoE is to keep the "Hot" layers (Attention, Norms) on GPU and offload the massive "Cold" layers (The Experts/FFN).
- **Command:**

./llama-server... -ngl 99 -ot ".*ffn*." ``` The -ot (Offload Tensors) flag allows regex-based targeting. .*ffn.* targets the Feed-Forward Networks (where the experts live) to be placed on the CPU. This allows the GPU to handle the attention mechanism (speeding up context) while the CPU manages the bulk storage of expert weights.[17]

6.4 The --swa-full Flag (Alternative Solution)

If metadata patching is not possible, the --swa-full flag attempts to force llama.cpp to read the full context window regardless of the model's sliding window parameter.

- Command: ./llama-server... --swa-full
- Effectiveness: This flag was introduced specifically to address SWA limitations.[7] However, user reports suggest it may not always override the *initialization check* for cache reuse. It is worth trying as a first non-destructive step. If the log still says "cache_reuse disabled", proceed to GGUF patching.

---

7. Performance Modeling & Hardware Implications

7.1 Throughput Analysis

Let us quantify the impact of the fix.

Assume a prompt of 4,000 tokens (T_{prompt}) and a generation of 100 tokens (T_{gen}).

Assume hardware processing speeds:

- Prefill Speed (S_{pre}): 30 tokens/sec (MoE models are slow at prefill due to weight loading).
- Generation Speed (S_{gen}): 10 tokens/sec.

Scenario A: Without Cache Reuse (Current State)

For every turn, we process $T_{prompt} + T_{gen}$.

$$Time = \frac{4000}{30} + \frac{100}{10} = 133.3 + 10 = \mathbf{143.3} \text{ seconds}$$

Scenario B: With Cache Reuse (Patched State)

We reuse 3,950 tokens. We verify/process 50 new tokens (T_{new}) + generation.

$$Time = \frac{50}{30} + \frac{100}{10} = 1.66 + 10 = \mathbf{11.66} \text{ seconds}$$

Result: A **12x** improvement in response latency. This difference defines whether the model is usable for interactive chat or not.

7.2 VRAM Requirements Table

Based on [18] and internal calculations, here is the VRAM footprint for gpt-oss-120b under different configurations.

Component	Specification	Size (Est.)
Model Weights	MXFP4 Quantization	59.0 GB
KV Cache (32k)	Q8_0	~2.4 GB
KV Cache (128k)	Q8_0	~9.6 GB
KV Cache (128k)	FP16	~19.2 GB
Compute Buffer	Flash Attention Overhead	~1-2 GB
Total (Minimal)	32k Ctx, Q8 Cache	**~63 GB**
Total (Max)	128k Ctx, FP16 Cache	**~80 GB**

Hardware Recommendation:

- **Minimum:** 2x RTX 3090/4090 (48GB) + 64GB System RAM (Offloading required).
- **Optimal:** 1x A100 80GB or 3x 3090/4090 (72GB) or Mac Studio M2/M3 Ultra (128GB Unified Memory).
- **Note:** For the user's specific context (59GB model), a purely GPU-resident setup requires at least 80GB of VRAM (e.g., A100 or multi-GPU). With less, the -ot CPU offloading strategy is mandatory.

---

8. Alternative Approaches & Future Roadmap

8.1 Comparison with vLLM

Snippet [19] mentions vllm performance on gpt-oss. vllm is an alternative inference engine focused on high-throughput serving using PagedAttention.

- **Pros:** vllm handles memory fragmentation better and may support SWA natively without disabling cache reuse (using block tables).
- **Cons:** vllm typically requires Nvidia GPUs and does not support the broad range of quantization (GGUF/legacy) and hybrid offloading (CPU/GPU) that llama.cpp excels at. For a user on Apple Silicon or using consumer GPUs with system RAM offloading, llama.cpp remains the only viable option.

8.2 Upcoming Llama.cpp Features

The llama.cpp development roadmap includes "Ring Buffer" support for KV cache.

- **Current State:** The conflict exists because the ring buffer logic is not fully integrated with the sequence-based prefix matching.
- **Future State:** Developers are working on mapping virtual token positions to physical ring slots, which would allow SWA *and* Cache Reuse to coexist.[20]
- **Recommendation:** Until this feature lands in a stable build, the GGUF metadata patch is the definitive fix.

---

9. Conclusion and Action Plan

The inability to reuse the KV cache with gpt-oss-120b in llama.cpp is a solvable configuration conflict, not a fundamental model defect. By understanding that the engine's safety checks effectively quarantine the model due to its sliding_window metadata, we can bypass the restriction.

Final Recommendations:

1. **Patch the Model:** Use gguf-new-metadata.py to remove gpt-oss.attention.sliding_window from the GGUF file. This is the "Silver Bullet" solution.
2. **Optimize the Runtime:**
   - Use --cache-reuse 256 to stabilize the reuse threshold.
   - Use -ctk q8_0 -ctv q8_0 to fit the 128k context into VRAM.
   - Use -fa (Flash Attention) to ensure computational viability at long contexts.
   - Use -ot ".*ffn.*" if VRAM is insufficient to hold the full 59GB weights + 10GB cache.
3. **Monitor Logs:** Verify the fix by watching for kv cache reuse or stable n_past values in the server logs, ensuring the `kv cache rm (GitHub Discussion #15986)[2] (GitHub Issue #15894 - Log analysis).

- **GGUF Structure:** [5] (Metadata Dump)[13] (GGUF Script Documentation).
- **Architecture:** [6] (MoE, YaRN, SWA details).
- **Configuration Flags:** [10] (Cache reuse CLI)[7] (SWA Full flag)[17] (Offloading strategies).
- **Hardware Benchmarks:** [18] (VRAM Usage)[23] (Multi-GPU setups).

(End of Report)

Works cited

1. Why Does Inference Speed Collapse After a Few Rounds of Conversation? Tried Cache-Reuse and Keep — Any Fixes? · ggml-org llama.cpp · Discussion #15986 - GitHub, accessed January 26, 2026, https://github.com/ggml-org/llama.cpp/discussions/15986

2. Eval bug: gpt-oss model reprocess the entire prompt from beginning. · Issue #15894 · ggml-org/llama.cpp - GitHub, accessed January 26, 2026, https://github.com/ggml-org/llama.cpp/issues/15894
3. PSA/RFC: KV Cache quantization forces excess processing onto CPU in llama.cpp - Reddit, accessed January 26, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1ng0fmv/psarfc_kv_cache_quantization_forces_excess/
4. Anyone actully try to run gpt-oss-120b (or 20b) on a Ryzen AI Max+ 395? : r/LocalLLaMA - Reddit, accessed January 26, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1nabcek/anyone_actully_try_to_run_gptoss120b_or_20b_on_a/
5. GPU-Accelerated LLM Setup: gpt-oss-20b & llama-cpp - Kaggle, accessed January 26, 2026, https://www.kaggle.com/code/ismailkm/gpu-accelerated-llm-setup-gpt-oss-20b-llama-cpp
6. OpenAI's Open-Weight Models: A Technical Deep Dive into the gpt-oss-120b and gpt-oss-20b Models - Shubham, accessed January 26, 2026, https://shubh7.medium.com/openais-open-weight-models-a-technical-deep-dive-into-the-gpt-oss-120b-and-gpt-oss-20b-models-0e831def0e2f
7. Misc. bug: --cache-reuse no longer seems to be caching prompt prefixes · Issue #15082 · ggml-org/llama.cpp - GitHub, accessed January 26, 2026, https://github.com/ggml-org/llama.cpp/issues/15082
8. How Attention Sinks Keep Language Models Stable : r/LocalLLaMA - Reddit, accessed January 26, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1mkvks4/how_attention_sinks_keep_language_models_stable/
9. Server: reuse cached tokens for shifted prompt · Issue #5793 · ggml-org/llama.cpp - GitHub, accessed January 26, 2026, https://github.com/ggml-org/llama.cpp/issues/5793
10. llama.cpp not using kv cache effectively? : r/LocalLLaMA - Reddit, accessed January 26, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1kkocfx/llamacpp_not_using_kv_cache_effectively/
11. Sliding Window Attention support merged into llama.cpp, dramatically reducing the memory requirements for running Gemma 3 : r/LocalLLaMA - Reddit, accessed January 26, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1kqye2t/sliding_window_attention_support_merged_into/
12. Best recommendation/explanation for command for llama.cpp for oss-gpt 120b? - Reddit, accessed January 26, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1n8j2tu/best_recommendation_explanation_for_command_for/
13. llama.cpp/gguf-py/README.md at master · ggml-org/llama.cpp · GitHub, accessed January 26, 2026, https://github.com/ggml-org/llama.cpp/blob/master/gguf-py/README.md

14. gguf-new-metadata(1) - Debian experimental, accessed January 26, 2026, https://manpages.debian.org/experimental/python3-gguf/gguf-new-metadata.1.en.html
15. Why is the context (KV cache) vram amount for gpt-oss 120b so low : r/LocalLLaMA - Reddit, accessed January 26, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1oqnr30/why_is_the_context_kv_cache_vram_amount_for/
16. How to run LLMs on PC at home using Llama.cpp - The Register, accessed January 26, 2026, https://www.theregister.com/2025/08/24/llama_cpp_hands_on/
17. gpt-oss: How to Run Guide | Unsloth Documentation, accessed January 26, 2026, https://unsloth.ai/docs/models/gpt-oss-how-to-run-and-fine-tune
18. GPT-OSS 120B: Specifications and GPU VRAM Requirements - ApX Machine Learning, accessed January 26, 2026, https://apxml.com/models/gpt-oss-120b
19. vLLM on GB10: gpt-oss-120b MXFP4 slower than SGLang/llama.cpp... what's missing?, accessed January 26, 2026, https://forums.developer.nvidia.com/t/vllm-on-gb10-gpt-oss-120b-mxfp4-slower-than-sglang-llama-cpp-what-s-missing/356651
20. PR #11213 llama : refactor llama_kv_cache, llama_context and llm_build_context, accessed January 26, 2026, https://app.semanticdiff.com/gh/ggerganov/llama.cpp/pull/11213/overview
21. gguf-set-metadata.py - Llama.Cpp - GitLab, accessed January 26, 2026, https://gitlab.informatik.uni-halle.de/ambcj/llama.cpp/-/blob/b2001/gguf-py/scripts/gguf-set-metadata.py
22. OpenAI GPT-OSS: A Detailed Guide and Implementation | by Sai Dheeraj Gummadi, accessed January 26, 2026, https://medium.com/data-science-in-your-pocket/openai-gpt-oss-a-detailed-guide-to-the-new-era-of-open-source-ai-b6e4200f12aa
23. guide : running gpt-oss with llama.cpp #15396 - GitHub, accessed January 26, 2026, https://github.com/ggml-org/llama.cpp/discussions/15396