llama.cpp RPC Multi-GPU Setup Guide (Windows 11 & Linux Distributed Cluster)

This was generated by NOTEBOOKLM from a lot of sources.

Question

I have three computers at home. one AMX Ryzen 7800 X3D with 32GB RAM, NVIDIA RTX 5070 12GB VRAM and windows 11. Two machines with an intel i5 11th gen, 16GB RAM and NVIDIA RTX 2070 tier 8GB VRAM.

Show or create a document which gives me extract instructions on how to setup llama rpc utilizing all three machines. which OS should I install on the two smaller intel i5 machines. how do i setup the llama rpc to utilize the total of 28GB VRAM for a suitable model.

Prompt used by NotenookLM

A technical guide with exact instructions on setting up llama.cpp RPC across one Windows 11 host and two Linux worker nodes, pooling 28GB of VRAM.

Answer

This technical deployment guide details the configuration of a high-performance distributed inference cluster. We will shard a Large Language Model (LLM) across a heterogeneous environment comprising one Windows 11 Master node and two Linux Worker nodes, pooling a total of 28GB of VRAM.

System Architecture & VRAM Pooling Strategy

In a distributed RPC (Remote Procedure Call) architecture, the model is sharded across the network. The Windows 11 Master node acts as the orchestrator; it physically hosts the .gguf model file and manages the Key-Value (KV) cache. The Linux Worker nodes act as "virtual GPUs," receiving tensor calculation tasks over the network, executing them on local hardware, and returning the results.

The Memory Math: 70B Parameter Constraints

Running a 70B model on a 28GB VRAM pool requires an aggressive offloading strategy. A standard 70B model quantized at Q4_K_M typically requires 42–43 GB just for the weights.

Component Memory Requirement (70B Q4_K_M) Pooled VRAM (28GB) VRAM Deficit (System RAM Usage) Model Weights ~43 GB 28 GB -15 GB KV Cache (8K Context) ~5 GB 0 GB -5 GB KV Cache (32K Context) ~25 GB 0 GB -25 GB Total Deficit 48–68 GB -- -20 to -40 GB

Architect's Note: Because the model weights (43GB) exceed your physical VRAM pool (28GB), approximately 15GB of weights—plus the entirety of the KV cache—will reside in the Master node's system RAM. Expect a performance ceiling of 8–12 tokens per second (TPS) for a single stream, heavily dependent on the Master node's CPU and memory bandwidth.

The Memory Bandwidth Lever

During the decode phase (token generation), performance is strictly memory-bound. While the prefill phase (prompt processing) is compute-heavy, generation speed scales with how fast weights can be streamed from memory. In this distributed setup, your primary performance bottleneck is the movement of data between the system RAM, the local GPU, and the remote RPC workers.

Environment Preparation & Building llama.cpp

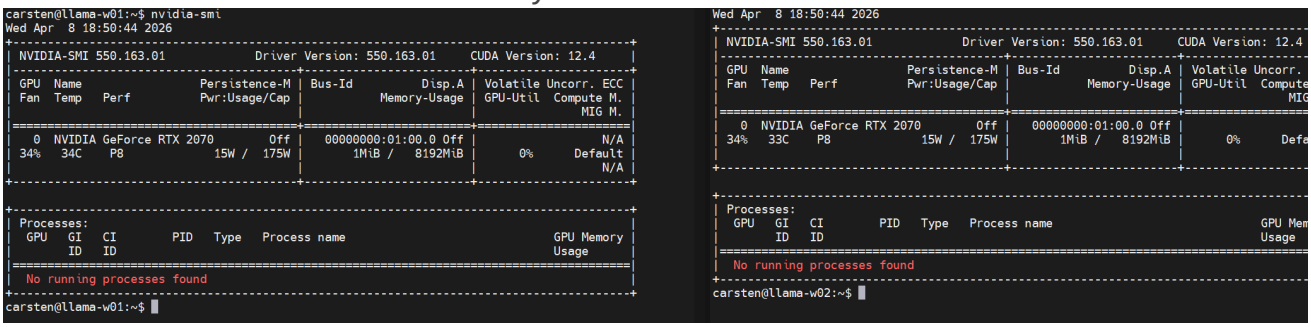
See also: https://linuxconfig.org/debian-13-nvidia-driver-installation

Linux Worker Nodes (Ubuntu/Debian)

Ensure both worker nodes are updated and have the NVIDIA stack correctly initialized.

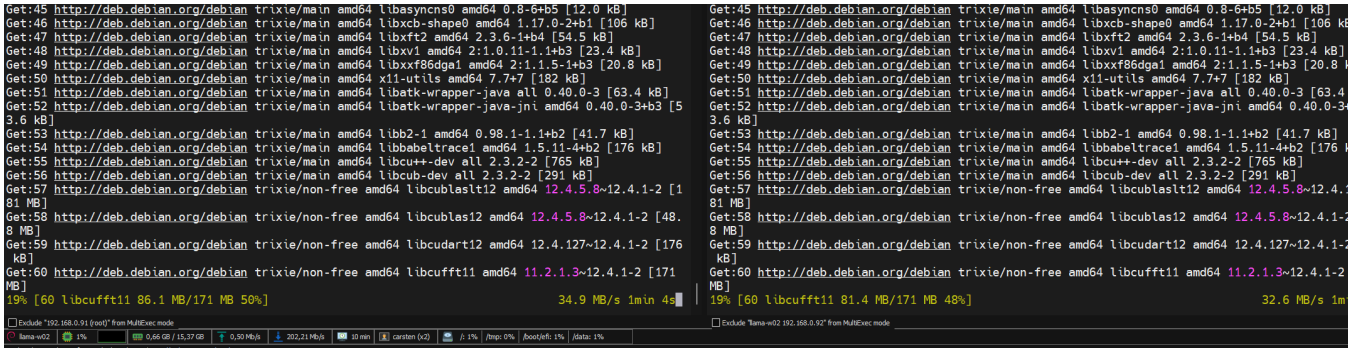
1. Toolchain: `sudo apt install build-essential g++ cmake git curl libcurl4-openssl-dev pciutils`
2. Install NVIDIA Drivers

- Update APT Sources List with non-free
- `sudo sed -i 's/main/main non-free contrib/g' /etc/apt/sources.list`
- `sudo apt update`
- Next install the required header
- `sudo apt install linux-headers-$(uname -r) build-essential dkms nvidia-detect`
- run `nvidia-detect` to check if GPU is detected
- If card detected, install drivers
- `sudo apt install nvidia-driver nvidia-kernel-dkms`
- Select OK when asked about the "wrong" kernel.
- `sudo reboot`
- After reboot run `nvidia-smi` to verify



3. CUDA Toolkit: Install nvidia-cuda-toolkit (required for the NVCC compiler).

   `sudo apt install nvidia-cuda-toolkit`



4. Compute Capability: Identify your GPU's architecture:
   `nvidia-smi --query-gpu=compute_cap --format=csv`

Windows 11 Master Node

The Master node must run a version of llama-cli.exe that matches the build version of the Linux workers exactly.

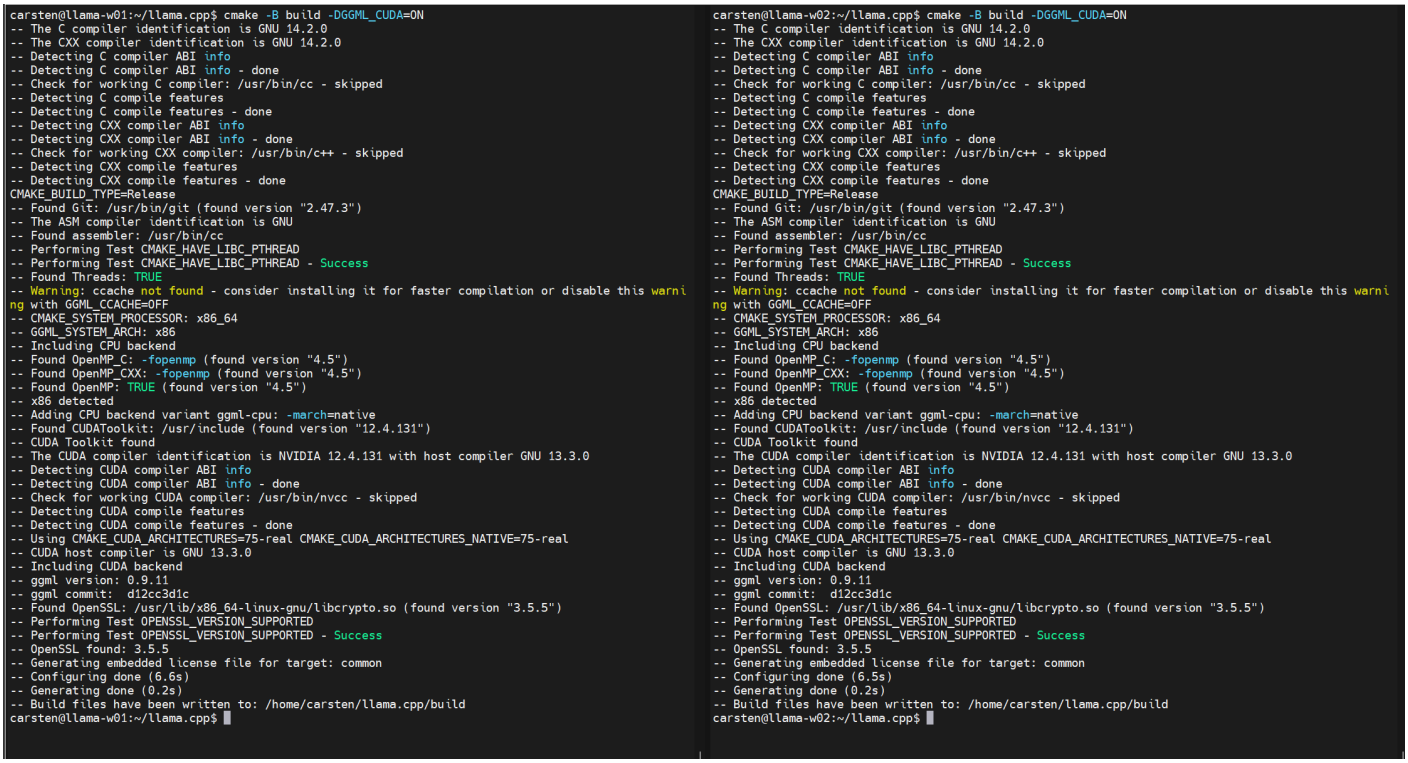
- Version Sync: Ensure all nodes are built from the same Git commit to avoid RPC protocol mismatches.
- Build Method: It is recommended to use the pre-compiled Windows releases with CUDA support or build via CMake to ensure llama-cli.exe is optimized for your local GPU.

Download and build LLAMA.cpp

https://github.com/ggml-org/llama.cpp

```
git clone https://github.com/ggml-org/llama.cpp
cd llama.cpp

# following the CUDO build here https://github.com/ggml-org/llama.cpp/blob/master/docs/build.md#cuda
cmake -B build -DGGML_CUDA=ON
```



Next build the tool

```
# Use j parameter to have multiple jobs parallel
# It will speed things up significantly.
# Put the number of your available cores in
cmake --build build --config Release -j 8
```

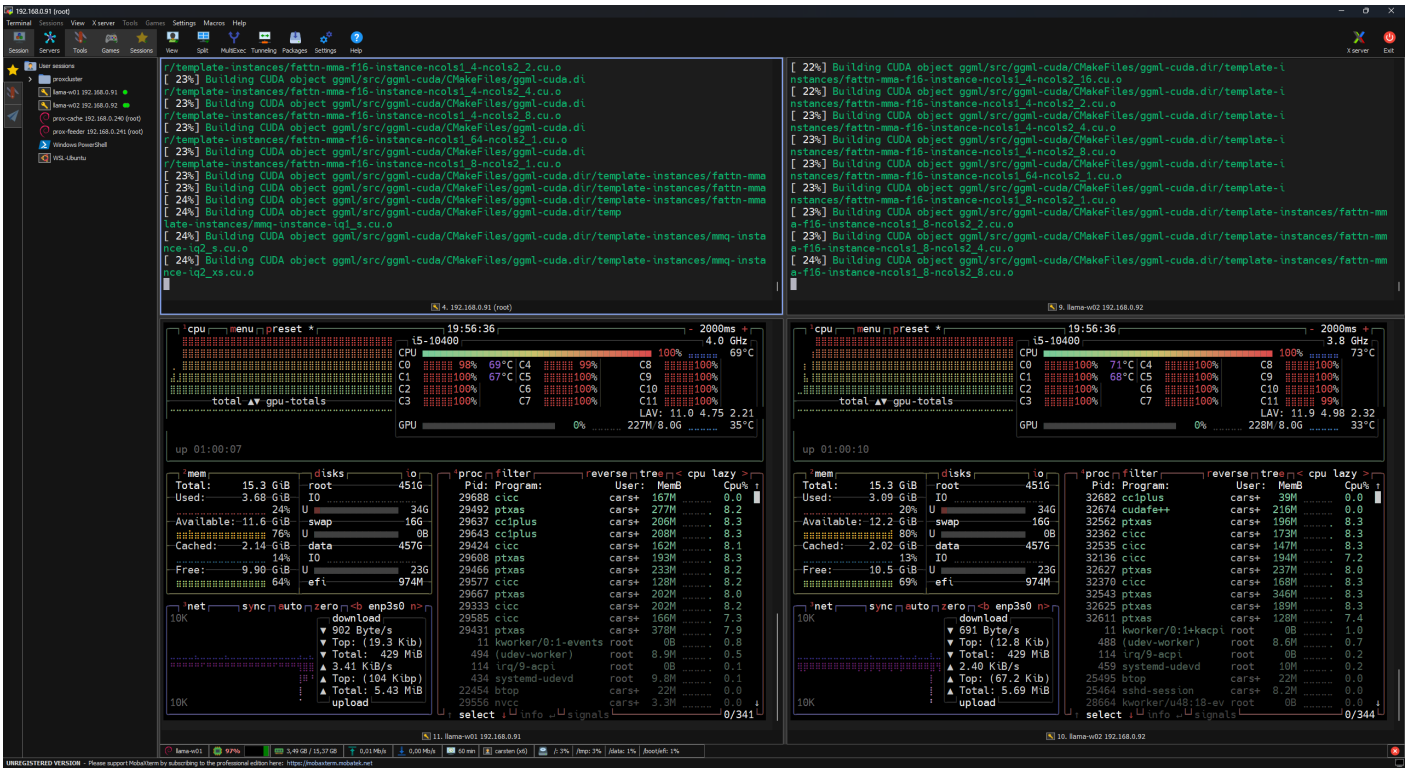
Running `btop` in another terminal to see if something is still happening.

Compiling again with the RPC=ON flag following these instructions: https://github.com/ggml-org/llama.cpp/tree/master/tools/rpc#remote-hosts

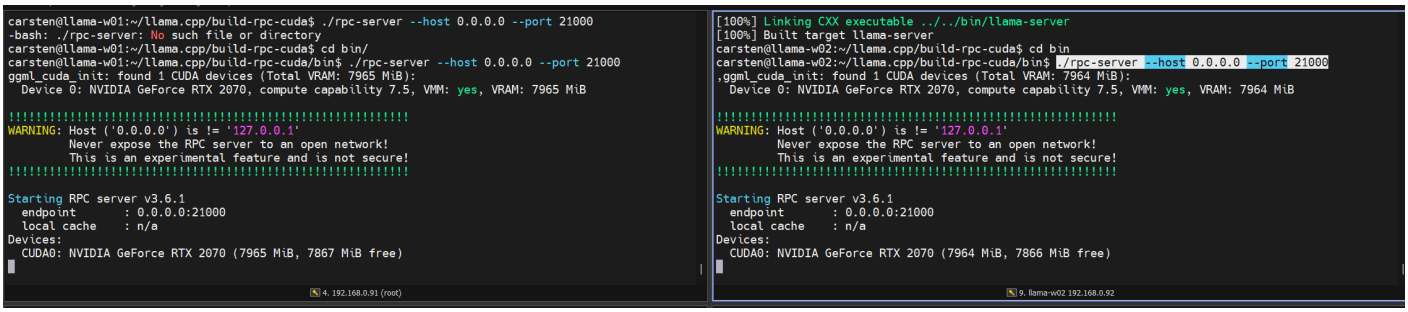
In the llama.cpp directory (the git clone) run the following cmds

```
mkdir build-rpc-cuda
cd build-rpc-cuda
cmake .. -DGGML_CUDA=ON -DGGML_RPC=ON
cmake --build . --config Release
```

This will build the rpc-server executable that we need.



After the build finished, we can change into the bin folder and run the rpc-server



Download and build LLAMA.cpp on Windows

Install with WINGET

Networking & Security Requirements

Do not use Wi-Fi. Distributed inference is highly sensitive to network latency. Wi-Fi introduces jitter and high latency that will degrade token generation to unusable speeds. Wired Gigabit Ethernet is the mandatory baseline for a stable RPC cluster.

Security Protocols

The llama.cpp RPC protocol transmits unencrypted model tensors and hidden states. Exposing these ports to the WAN is a critical security vulnerability.

- Internal Only: Restrict RPC traffic to trusted internal IPs or a secured VPC.
- Firewall: Ensure the chosen ports (default 50052 or custom) are open on Linux workers for the Master's IP.

Linux Worker Configuration (RPC Server)

Each worker creates a socket to listen for incoming data (model parameters and hidden states) from the master.

Command to initialize:

./rpc-server --host 0.0.0.0 --port [PORT]

The --host 0.0.0.0 flag allows the server to listen on all interfaces, which is safe only if your local firewall/security groups are configured to restrict access.

---

5. Windows 11 Master Node Configuration

The Master node orchestrates the sharding by treating remote nodes as additional CUDA devices.

Launching Distributed Inference:

./llama-cli.exe -m llama-3-70b-q4_k_m.gguf --rpc [WORKER_1_IP]:[PORT],[WORKER_2_IP]:[PORT] --n-gpu-layers 100 --ctx-size 8192

- --rpc: Points to the remote "virtual GPUs."

- --n-gpu-layers 100: High value ensures the engine attempts to offload every layer possible into the pooled VRAM before falling back to system RAM.

---

6. Performance Tuning & Optimization

To maximize tokens per second in an offloading scenario, use the following "Quick Win" flags:

- --flash-attn: Enables Flash Attention to reduce the memory footprint and compute time of the attention mechanism.
- --mlock: Force-locks the model in RAM, preventing the operating system from swapping weights to the disk (pagefile), which is a common cause of stuttering.
- --no-mmap: Recommended for the worker nodes if they have fast NVMe drives but limited system RAM; it forces a full load of the shard into memory at startup.
- --threads: Match the thread count to the physical CPU cores for the Master's fallback processing.

For GPU-assisted or distributed inference, the optimal thread count is typically 4–8 threads, regardless of the total core count. Over-allocating threads often leads to context-switching overhead that reduces performance.

---

7. Troubleshooting & Connectivity Diagnostics

Error: "RPC Failed to Connect" or "0 MiB free"

If the Master connects but the worker reports "0 MiB free" on the device:

1. Backend Failure: This usually indicates the RPC server is reachable, but the CUDA backend failed to initialize the GPU.
2. Arch Mismatch: Check if CUDA_DOCKER_ARCH was set correctly during the build on the Linux worker.
3. Toolkit Issue: Ensure the nvidia-cuda-toolkit is properly installed and nvidia-smi shows no errors.

Version Mismatch

If the nodes connect but crash during tensor loading, verify that the llama.cpp version (Git hash) is identical on Windows and Linux. RPC is under active development and protocol changes are frequent.

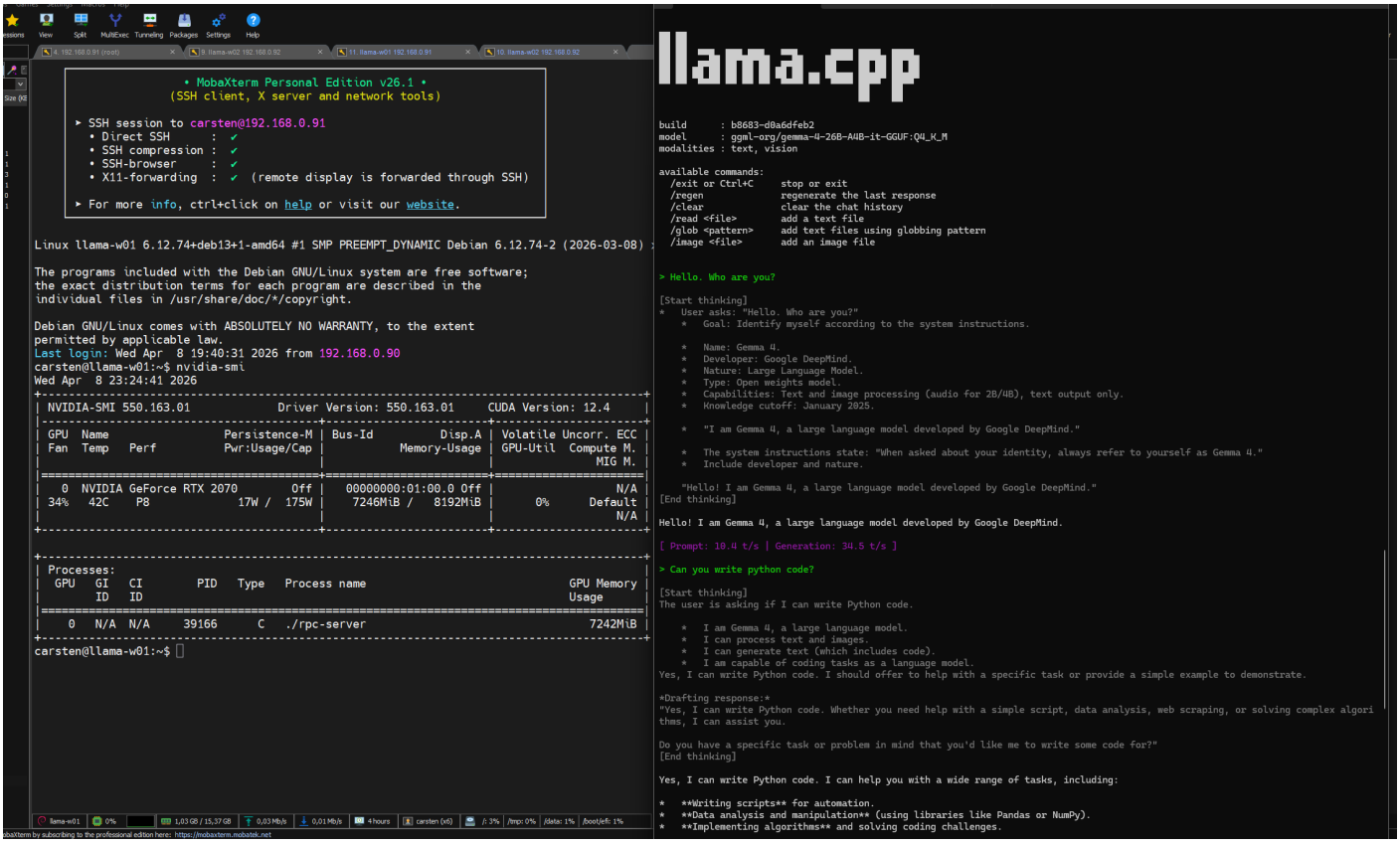
---

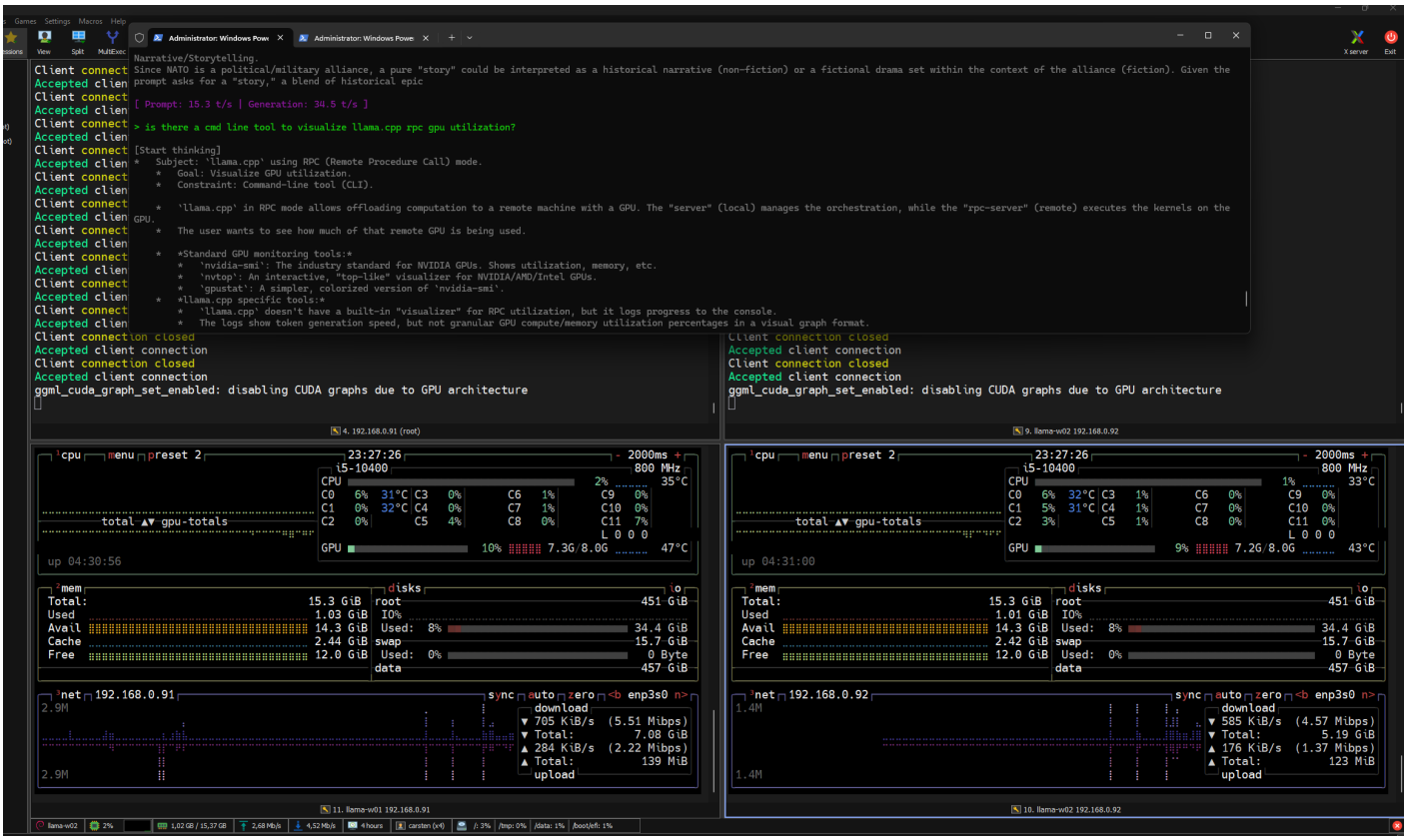
8. Advancing the Setup: RAG Integration

Because the 28GB pool is insufficient for the weights and KV cache of a 70B model, Retrieval-Augmented Generation (RAG) is the superior strategy for handling large datasets.

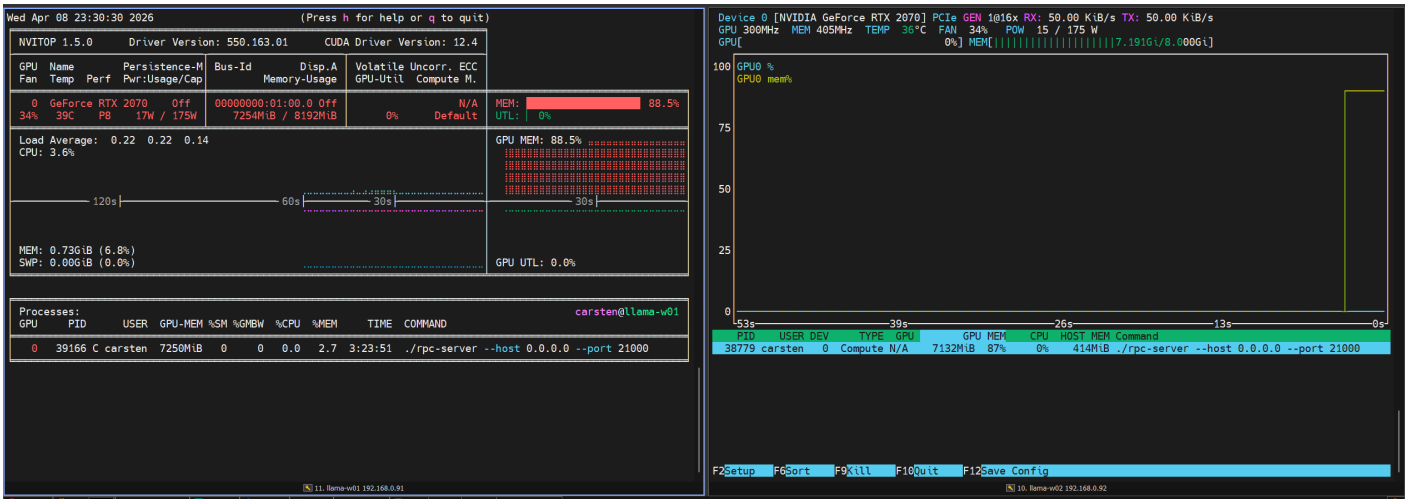
Instead of extending the context window (which exponentially consumes VRAM and increases latency), RAG pulls relevant text chunks from an external database only when needed. This keeps your context window small (e.g., 8K), preserves VRAM for the model weights, and prevents "Out of Memory" (OOM) errors while maintaining high accuracy for documentation-heavy tasks.

Some Screenshots

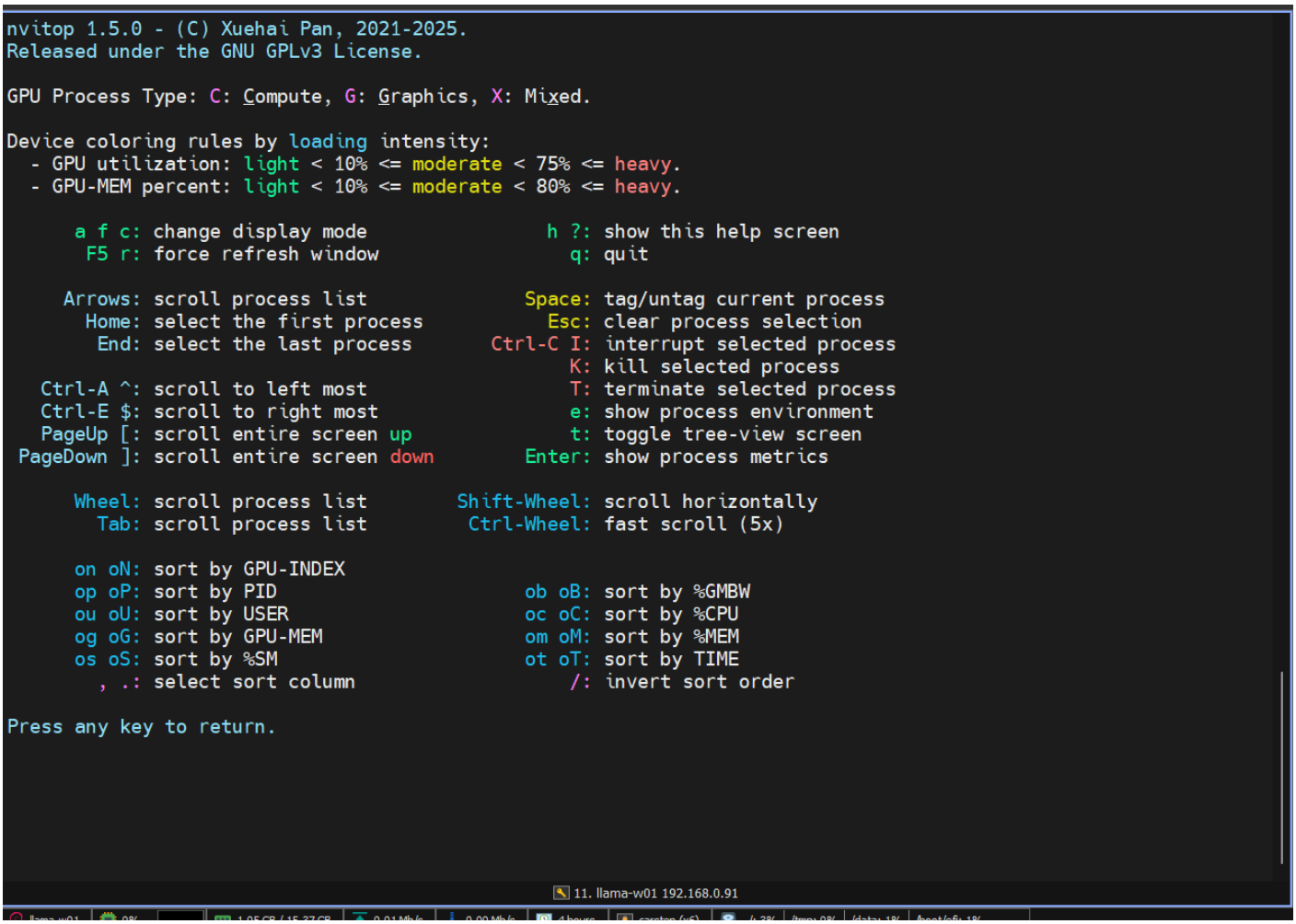




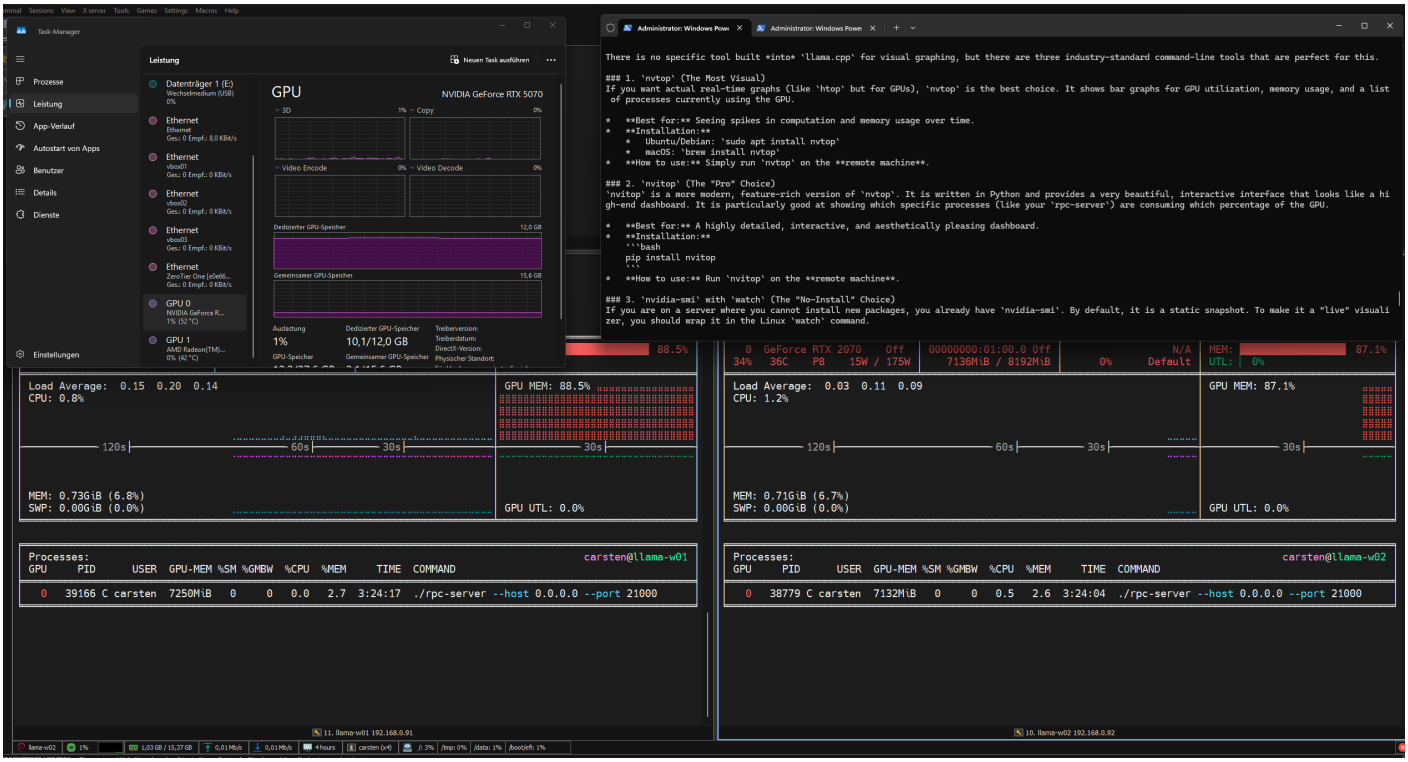
NVTOP



NVITOP



NVITOP, NVTOP and Terminal with llama-cli



After running

```
Exiting...
llama_memory_breakdown_print: | memory breakdown [MiB]        | total   free    self   model   context   compute    unaccounted |
llama_memory_breakdown_print: |   - RPC0 (192.168.0.91:21000) |  7965 =  705 + (7120 =  5168 +    1132 +     819) +         139 |
llama_memory_breakdown_print: |   - RPC1 (192.168.0.92:21000) |  7964 =  822 + (7002 =  4517 +    2132 +     353) +         138 |
llama_memory_breakdown_print: |   - Vulkan0 (RTX 5070)        | 11855 =    2 + (8994 =  6316 +    2156 +     521) +        2858 |
llama_memory_breakdown_print: |   - Host                      |                 1103 =   577 +       0 +     526                |
PS C:\Users\Admin>
```

Revision #8

Created 2026-04-08 12:36:07 UTC by Carsten

Updated 2026-04-08 21:36:26 UTC by Carsten