Optimization of Windows 11 24H2 and WSL2 for High-Performance AI Workloads on Constrained Hardware

1. Executive Introduction: The Convergence of Consumer Hardware and AI Architectures

The year 2025 marks a pivotal transition in the landscape of local artificial intelligence development. The democratization of Large Language Models (LLMs), diffusion models, and complex neural networks has shifted the computational locus from centralized cloud clusters to local workstations. However, this shift has exposed a significant disparity between the hardware requirements of modern AI architectures and the specifications of standard consumer electronics. A particularly acute manifestation of this challenge is the attempt to run substantial AI workloads—such as quantized LLMs (e.g., LLaMA-3 variants), agentic browsers like "Atlas," or deep learning training loops—on systems limited to 16GB of Random Access Memory (RAM).

In this constrained environment, the operating system (OS) ceases to be merely a platform and becomes a competitor for resources. Windows 11, particularly the 24H2 feature update, introduces sophisticated capabilities such as "Mirrored Networking" and "Auto Memory Reclaim" that theoretically enhance the virtualization experience.[1] Yet, the inherent overhead of the Windows kernel, combined with the memory-intensive nature of virtualization via the Windows Subsystem for Linux version 2 (WSL2), creates a zero-sum game. Every megabyte claimed by a background Windows service is a megabyte denied to the CUDA tensors residing in the GPU's shared memory fallback or the Linux page cache.

This technical monograph provides an exhaustive analysis of optimizing a Windows 11 24H2 host to support AI workloads within WSL2 on a 16GB physical memory footprint. The analysis assumes the utilization of "AtlasOS"—a specialized, debloated configuration of Windows 11—and explores the granular configuration of the Microsoft Hyper-V hypervisor to achieve a stable, high-performance compute node. The objective is to engineer a system state where approximately 6GB of physical RAM is consistently available to the Linux guest for AI processing, while maintaining sufficient host stability for browser-based research and interface management.

1.1 The Architectural Challenge: Type-1 Hypervisors and Memory Pressure

To understand the optimization strategy, one must first deconstruct the virtualization architecture. WSL2 is not a translation layer (like its predecessor, WSL1); it utilizes a

lightweight utility Virtual Machine (VM) running a genuine Linux kernel. This VM sits atop the Microsoft Hyper-V hypervisor.

In a standard deployment, Windows 11 operates as the "root partition" or "host," retaining direct control over hardware management, while WSL2 operates as a "child partition." Both compete for the same pool of physical RAM.

- **The Windows Demand:** A standard Windows 11 24H2 installation, laden with telemetry, prefetching (SysMain), and integrated services (Teams, OneDrive, Copilot), typically establishes a "committed memory" floor of 4GB to 8GB at idle.[3]
- **The Linux Demand:** The AI workload within WSL2 requires memory for the kernel, the container runtime (Docker or Podman), the Python runtime, and the model weights themselves. Crucially, Linux utilizes unused RAM for caching disk I/O (Page Cache), which speeds up data loading for training.

The conflict arises because the Hyper-V hypervisor sees the Linux Page Cache as "active" memory. Without intervention, the WSL2 VM will grow to consume 50% or 80% of the host RAM [5], potentially starving the Windows host and causing system instability or triggering the Windows Memory Manager to aggressively swap pages to the SSD. The optimization of this stack requires a holistic approach: aggressive reduction of the Windows host footprint via AtlasOS techniques, precise configuration of the hypervisor's memory boundaries via .wslconfig, and the selection of efficient container runtimes.

---

2. The Host Environment: Windows 11 24H2 and AtlasOS Integration

The foundation of a high-performance AI workstation on 16GB hardware is the minimization of the host operating system's resource consumption. If the host OS consumes 50% of the available memory, the "6GB usable" goal for the guest becomes mathematically impossible without inducing severe thrashing (swapping to disk).

2.1 The AtlasOS Paradigm: Deconstructing Windows 11

"AtlasOS" represents a community-driven methodology for modifying the Windows installation image or post-installation state to remove non-essential components.[6] In the context of 2025, specifically for AI workloads, the value proposition of AtlasOS shifts from "gaming frames per second" to "available memory pages."

2.1.1 Version Compatibility and Deployment Strategy

As of January 2025, the stable target for AI development is Windows 11 24H2. While newer "Insider" builds (25H2) exist, research indicates that AtlasOS playbooks often lag slightly

behind the bleeding edge of Microsoft's release cadence.[7]

- **Compatibility Matrix:** Attempting to apply AtlasOS playbooks designed for 23H2 or early 24H2 to a 25H2 build results in "Playbook not supported" errors or unstable system states. The recommendation is to perform a clean installation of Windows 11 24H2 and apply the corresponding verified AtlasOS playbook.
- **The Debloating Mechanism:** AtlasOS modifies the Windows image to remove high-overhead components such as:
    - **Microsoft Defender:** (Optional but common in Atlas) Removing the real-time scanning engine (MsMpEng.exe) can free 200MB-500MB of RAM and significantly reduce CPU latency during the loading of large datasets (e.g., thousands of image files for training). *Note: This requires the machine to be air-gapped or behind a strict hardware firewall for security.*
    - **Telemetry Agents:** Disabling the "Connected User Experiences and Telemetry" service (DiagTrack) prevents the periodic transmission of usage data, which saves memory and reduces background network activity—a critical factor when mitigating the NDU memory leak discussed later.[9]
    - **AppX Deployment:** Removal of pre-installed Store apps (Clipchamp, Solitaire, etc.) reduces the number of dormant processes.

2.2 Deep Dive: Windows Service Optimization

Beyond the broad strokes of AtlasOS, specific Windows services must be manually tuned to accommodate the unique I/O patterns of AI workloads.

2.2.1 SysMain (SuperFetch) and the SSD Dichotomy

The SysMain service is designed to analyze usage patterns and preload frequently used applications into the "Standby" list of the system RAM. On a general-purpose PC with an HDD, this is beneficial. On a 16GB AI workstation with an NVMe SSD, it is detrimental.

- **The Conflict:** AI workloads are characterized by massive, bursty memory allocations (loading a 6GB model). If SysMain has filled 4GB of RAM with preloaded browser libraries and shell extensions, the Memory Manager must "zero out" those pages before they can be assigned to WSL2. This introduces latency. Furthermore, SysMain competes for the very limited physical RAM that we need to dedicate to the vmmem process.
- **2025 Status:** Research confirms that on modern SSDs, the launch time penalty of disabling SysMain is imperceptible (milliseconds), while the memory consistency gain is significant.[10]
- **Configuration:**
    - **PowerShell Command:** Set-Service -Name "SysMain" -StartupType Disabled; Stop-Service -Name "SysMain" -Force
    - **Impact:** Ensures that "Free" memory remains truly free for allocation to the Hyper-V child partition.

2.2.2 The NDU Memory Leak: A Persistent Threat

The Windows Network Data Usage (NDU) monitoring driver (Ndu.sys) has a documented history of causing non-paged pool memory leaks. This occurs when the driver allocates memory to track network flows but fails to release it properly, particularly under high-throughput scenarios.

- **Relevance to AI:** AI workflows often involve pulling massive Docker images (e.g., pytorch/pytorch:latest can be >8GB) or downloading datasets from Hugging Face. This generates high-throughput network traffic.
- **Current Status (2024-2025):** Despite multiple Windows updates, user telemetry and reports in 2025 indicate that the NDU leak persists on specific hardware configurations.[12] The symptom is a slow, creeping increase in RAM usage that is not attributed to any specific process in Task Manager, eventually forcing a reboot.
- **The Fix:** The registry modification to disable the NDU driver remains the gold standard for stability.
  - **Registry Path:** HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Ndu
  - **Value:** Change Start to 4 (Disabled).
  - **Consequence:** The "Data Usage" statistics in Windows Settings will no longer function, a negligible loss for a development workstation.

2.2.3 AppX Deployment Service (Appxsvc)

In late 2025 builds of Windows 11 (24H2/25H2), Microsoft changed the startup type of Appxsvc from "Manual" to "Automatic".[14] This service manages Microsoft Store apps.

- **Optimization Strategy:** If the Linux distribution (Ubuntu) was installed via the Microsoft Store, this service *must* remain enabled to facilitate updates and licensing checks. However, if the user installs WSL2 distros via the wsl --import command using a root filesystem tarball (bypassing the Store), Appxsvc can be safely disabled to reclaim further resources. For an AtlasOS setup, which often removes the Store entirely, ensuring this service is disabled prevents it from fruitlessly searching for updates.

2.2.4 Windows Search Indexing

Indexing is I/O and memory intensive. AI projects often contain thousands of small source files or data samples. The Windows Search Indexer (SearchIndexer.exe) will attempt to crawl the \\wsl$\Ubuntu\home... network share if mapped.

- **Configuration:** Exclude the WSL2 network mapped drive from Windows Search, or disable the WSearch service entirely if local file search is not required.
  - **PowerShell:** Set-Service -Name "WSearch" -StartupType Disabled

---

3. The Guest Environment: WSL2 Configuration and

.wslconfig

The .wslconfig file is the command center for the lightweight utility VM. Located in the user's home directory (%UserProfile%\.wslconfig), it dictates the resource boundaries and behaviors of the hypervisor. In January 2025, several features previously marked as "experimental" have matured into essential tools for memory management on constrained systems.

3.1 Memory Limits: The Hard Ceiling

By default, WSL2 in 24H2 attempts to allocate 50% or 80% of the host memory, depending on the specific build.[5] On a 16GB system, 50% (8GB) is a safe default, but 80% (12.8GB) risks starving the Windows host, leading to UI unresponsiveness.

- **The Strategy:** We must set a "Hard Ceiling" that guarantees the Windows host retains enough RAM to function (approx. 4GB for an optimized AtlasOS) while maximizing the allocation for AI.
- **Recommended Setting:** memory=8GB.
    - *Reasoning:* This allocates exactly 50% of the total RAM to the guest. This leaves 8GB for Windows. However, since we have optimized Windows to use only ~3GB, there is a 5GB buffer on the host side. Why not allocate more to WSL? Because the Windows kernel needs contiguous memory pages for device drivers (GPU) and high-performance I/O. Pushing the limit too high (e.g., 12GB) increases the risk of the host swapping, which degrades the performance of the entire system, including the VM.

3.2 Memory Reclamation: autoMemoryReclaim

This is arguably the most critical setting for 2025.

- **The Problem:** Linux uses free RAM as Page Cache. In earlier WSL2 versions, once Linux allocated RAM for cache, Hyper-V considered that RAM "used" and never reclaimed it. The vmmem process would grow to the 8GB limit and stay there, even if the Linux processes were idle.
- **The Solution:** The autoMemoryReclaim feature allows Hyper-V to reclaim cached memory from the guest VM and return it to the host pool.[1]
- **DropCache vs. Gradual:**
    - **dropCache:** This mode triggers a drop of the Linux caches immediately when the CPU is idle. Research [15] indicates this causes significant "stuttering" or "lag" in the system. The sudden purge of the cache forces the disk to wake up and re-read data when operations resume, creating latency spikes.
    - **gradual:** This mode uses a memory ballooning technique to slowly apply pressure to the Linux memory manager, forcing it to release the least-recently-used (LRU) cache pages over time. This is the optimal setting for AI workloads. It ensures that the "working set" of the AI model (the data currently being processed) remains in RAM,

while older cache data is returned to Windows.[17]
    - **Bug Watch:** Some users reported stuttering even with gradual in early builds. However, on 24H2 with the latest WSL kernel (5.15.167+), gradual is stable.

3.3 Disk Optimization: sparseVhd

The virtual hard disk (ext4.vhdx) used by WSL2 typically only grows in size. As AI datasets are downloaded, unpacked, and then deleted, the VHDX remains at its peak size on the host SSD.

- **The Feature:** sparseVhd=true.[1]
- **Mechanism:** When enabled, Hyper-V automatically punches holes in the VHDX file when blocks are freed inside the Linux filesystem. This reduces the disk footprint on the host. While primarily a storage optimization, it indirectly helps memory performance by reducing the strain on the host's filesystem driver during high I/O operations.

3.4 Networking Architecture: Mirrored Mode

Windows 11 24H2 introduced "Mirrored Networking" (networkingMode=mirrored), a fundamental shift from the legacy NAT approach.[17]

- **Legacy NAT:** WSL2 sits behind a virtual router. It has a separate IP. Accessing Windows apps requires host.docker.internal. IPv6 support is often broken or complex.
- **Mirrored Mode:** WSL2 shares the host's network interface. It has the same LAN IP. IPv6 works natively.
- **Relevance to AI:** Modern Docker containers and AI tools often assume IPv6 availability. Mirrored mode simplifies the networking stack, reducing the CPU overhead of packet translation (NAT).
- **Critical Dependency:** Mirrored mode relies on the **Host Network Service (HNS)**. In aggressive AtlasOS debloating scenarios, HNS might be disabled. If HNS is not running, WSL2 will fail to start network interfaces or fallback to a broken state.[18]
    - **Verification:** Get-Service hns in PowerShell. It must be Running.

3.5 The Optimized .wslconfig Artifact

Based on the synthesis of these factors, the following configuration file is mandated for the 16GB AtlasOS system.

Ini, TOML

```
# %UserProfile%\.wslconfig
[wsl2]
# MEMORY: Hard limit. 8GB allocated to guest.
```

```
# 6GB effectively usable for applications, 2GB for guest kernel/cache.
memory=8GB

# PROCESSORS: AI preprocessing (tokenization) is CPU bound.
# Expose all physical threads. Windows scheduler manages contention.
processors=8

# SWAP: Essential safety net. If PyTorch exceeds 8GB RAM,
# it swaps to this virtual disk instead of crashing via OOM Killer.
swap=8GB

# NETWORKING: Low latency, IPv6 support, direct LAN access.
networkingMode=mirrored
dnsTunneling=true
firewall=true
autoProxy=true

# EXPERIMENTAL FEATURES
[experimental]
# RECLAMATION: 'gradual' releases memory without causing stutter.
autoMemoryReclaim=gradual

# DISK: Keeps the VHDX file compact.
sparseVhd=true

# LOCALHOST: improved reliability for web-ui access.
hostAddressLoopback=true
```

---

4. The AI Subsystem: GPU Acceleration and Compute

The core of the user's requirement is the execution of "ATLAS" (AI workloads). This implies the utilization of NVIDIA CUDA. WSL2 achieves near-native GPU performance through **GPU Paravirtualization (GPU-PV)**.

4.1 The Driver Model: WDDM and GPU-PV

In the GPU-PV model, the Windows host runs the physical GPU driver (WDDM kernel mode driver). The Hyper-V hypervisor projects a virtual GPU device (/dev/dxg) into the Linux guest.

- **Driver Installation:** The user must install the **NVIDIA GeForce Game Ready Driver** or **Studio Driver** on *Windows*.
- **Prohibited Action:** The user must **NOT** install the NVIDIA driver (.run file or via apt install nvidia-driver-XXX) inside the WSL2 Linux instance. Doing so overwrites the libcuda.so

libraries provided by the Windows projection and breaks GPU access.[19]
- **Verification:** Running nvidia-smi inside WSL2 should return the GPU details immediately if the Windows driver is correctly installed.

4.2 Memory Hierarchies: VRAM, System RAM, and Shared Memory

AI models reside primarily in Video RAM (VRAM). However, when VRAM is exhausted (e.g., loading a 12GB model on an 8GB GPU), the system falls back to using System RAM. This is where the 16GB constraint becomes critical.

- **Shared Memory Fallback:** In 2025, the NVIDIA driver supports "System Memory Fallback." If the GPU needs more memory, it pages data over the PCIe bus to the system RAM allocated to WSL2.
- **The Bottleneck:** If .wslconfig limits WSL2 to 8GB, and the AI model tries to fall back 10GB of data, the process will crash.
- **PyTorch Shared Memory Crash:** A specific issue exists with PyTorch DataLoaders which use /dev/shm (shared memory) for Inter-Process Communication (IPC). By default, Docker containers often limit /dev/shm to 64MB. In WSL2, /dev/shm is usually half of the instance memory or a fixed size.
    - **Symptom:** RuntimeError: DataLoader worker (pid X) is killed by signal: Bus error.
    - **Fix:** When running containers (discussed in Section 6), explicitly set --shm-size=2gb or higher.
    - **NVIDIA Control Panel Tweak:** Snippet [20] suggests that for stability in mixed workloads, users can configure the "Sysmem Fallback Policy" in the NVIDIA Control Panel (Manage 3D Settings) to "Prefer No Sysmem Fallback" if they experience crashes. This forces the application to OOM (Out of Memory) gracefully rather than hanging the system by thrashing system RAM. However, for a 16GB system where we *want* to use RAM as overflow, the default policy is usually preferred *unless* specific instability is observed.

4.3 CUDA Support Status (2025)

As of January 2025, CUDA 12.x is fully supported in WSL2. PyTorch and ONNX Runtime with the CUDAExecutionProvider work out-of-the-box, provided the CUDA Toolkit (compiler and libraries) is installed in the Linux user space. The *driver* is provided by Windows, but the *toolkit* (nvcc) must be installed in Linux if compiling custom extensions.

---

5. Audio Subsystems: The WSLg Interface

If the "ATLAS" workload involves audio processing (e.g., speech recognition, text-to-speech, or voice agents), the audio subsystem becomes a critical component.

5.1 The Evolution: PulseAudio vs. PipeWire

- **Legacy (PulseAudio):** Historically, WSLg (the GUI architecture for WSL) utilized a PulseAudio server running on the Windows host to accept audio streams from Linux apps via a socket (/mnt/wslg/PulseServer).
- **Modern Standard (PipeWire):** In 2025, Linux distributions like Ubuntu 24.04 utilize PipeWire by default. PipeWire is a unified sound server that handles low-latency audio (replacing JACK) and consumer audio (replacing PulseAudio).[21]

5.2 The Conflict and Resolution in WSL2

A technical conflict arises because WSLg provides a PulseAudio socket, but the Ubuntu user space tries to start PipeWire.

- **Symptoms:** Audio devices missing in Linux applications, or high latency.
- **Resolution:** The recommended approach in 2025 is to utilize pipewire-pulse. This is a compatibility daemon that speaks the PulseAudio protocol but routes audio through the PipeWire graph.
    - **Configuration:** Users should ensure that the PULSE_SERVER environment variable points to the WSLg socket.
    - **Latency:** While PipeWire is low-latency, the transport over the RDP (Remote Desktop Protocol) rail used by WSLg introduces inevitable latency (typically 40-100ms). This is acceptable for playback but may be problematic for real-time monitoring.
    - **Microphone:** Microphone input in WSLg relies on the host exposing the recording device. Snippet [23] indicates reliability issues with USB microphones in this setup. For critical audio input, it is often more reliable to capture audio on the Windows side and stream it to the Linux AI service via a local TCP/UDP socket (e.g., using ffmpeg or a Python socket server) rather than relying on the WSLg hardware abstraction.

---

6. Container Orchestration: Docker vs. Podman

The choice of container runtime is the single most impactful software decision for memory optimization on a 16GB system.

6.1 Docker Desktop: The "Tax"

Docker Desktop is the industry standard, but it imposes a heavy resource tax.

- **Architecture:** Docker Desktop runs a dedicated Linux VM (even when using the WSL2 backend) or heavily utilizes the WSL2 utility VM with a complex shim process (com.docker.backend.exe). It also runs a system tray application and several background services on Windows.
- **Memory Cost:** The idle footprint of Docker Desktop on the Windows host is typically **500MB to 1GB**.[24]

- **Implication:** On a 16GB system targeting 6GB for AI, losing 1GB to the container orchestrator is suboptimal.

6.2 Podman: The Efficient Alternative

Podman (Pod Manager) is a daemonless container engine.

- **Architecture:** Podman uses a fork/exec model. The container process is a direct child of the Podman process. There is no central daemon running in the background with root privileges.
- **Memory Cost:** When no containers are running, Podman consumes **~0MB**. When a container runs, the overhead is minimal (just the process itself).
- **GPU Integration:** While Docker handles NVIDIA runtime setup automatically, Podman requires the **Container Device Interface (CDI)**.
  - **Setup:**
    1. Install nvidia-container-toolkit in WSL2.
    2. Run sudo nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml.
    3. This generates a spec file that allows Podman to discover the /dev/dxg device.
  - **Usage:** podman run --device nvidia.com/gpu=all...
- **Troubleshooting:** Snippet [25] highlights a common error: "Failed to initialize NVML". This is often due to a mismatch between the nvidia-container-toolkit version and the CDI spec version. Ensuring both are updated via the official NVIDIA repositories resolves this.

6.3 Recommendation

For this specific 16GB constrained environment, **Podman is the superior choice**. The reclamation of ~1GB of RAM outweighs the one-time configuration complexity of CDI.

---

7. Synthesis: The Realistic Budget and Implementation

Integrating the host optimization (AtlasOS), the guest configuration (.wslconfig), and the runtime choice (Podman), we can model the memory budget.

7.1 The "Realistic Budget" Table

Component	Standard Win 11 (24H2)	AtlasOS + Optimized WSL2	Gain
Windows Kernel & Drivers	4.5 GB	2.5 GB	+2.0 GB

Windows Background (Teams/OneDrive)	2.0 GB	0.2 GB	+1.8 GB
Container Engine (Idle)	0.8 GB (Docker)	0.0 GB (Podman)	+0.8 GB
Browser (Active Research)	1.5 GB	1.5 GB	0 GB
WSL2 VM Overhead	1.0 GB	0.5 GB	+0.5 GB
Total Overhead	**~9.8 GB**	**~4.7 GB**	**+5.1 GB**
RAM Available for AI Workload	**~6.2 GB (Unstable)**	**~11.3 GB (Theoretical)**	
Safe Limit (WSL2 Allocation)	**N/A**	**8.0 GB**	
Effective Usable AI Memory	**~4.0 GB**	**~6.5 GB**	**+2.5 GB**

Note: The "Effective Usable AI Memory" accounts for the Linux kernel overhead (~500MB) and the need for a safety buffer. With optimizations, allocating 8GB to WSL2 is safe, yielding roughly 6.5GB - 7.0GB for the actual PyTorch tensors.

7.2 Implementation Protocol

1. **Host Preparation:**
    - Install Windows 11 24H2.
    - Apply verified AtlasOS playbook.
    - **PowerShell:** Disable SysMain and NDU.
      ```
      PowerShell
      Set-Service -Name "SysMain" -StartupType Disabled
      Set-ItemProperty -Path "HKLM:\SYSTEM\ControlSet001\Services\Ndu" -Name "Start" -Value 4
      ```
    - Verify Host Network Service is running (Get-Service hns).

2. **WSL Configuration:**
    - Create %UserProfile%\.wslconfig with the optimized content (Section 3.5).
    - Run wsl --shutdown to apply changes.
3. **Runtime Setup:**
    - Install Ubuntu 24.04: wsl --install -d Ubuntu-24.04.
    - Remove Snap (optional, saves memory): sudo apt purge snapd.
    - Install Podman and NVIDIA Toolkit:
      ```
      Bash
      sudo apt-get update
      sudo apt-get install -y podman nvidia-container-toolkit
      sudo nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml
      ```

4. **Verification Steps:**
    - **Memory:** Open Windows Task Manager. Verify vmmem (WSL) process exists. Run a memory heavy task in Linux. Close it. Watch vmmem shrink slowly (proof of gradual reclaim).
    - **GPU:** Run podman run --rm --device nvidia.com/gpu=all docker.io/nvidia/cuda:12.3.1-base-ubuntu22.04 nvidia-smi.
    - **Audio:** Test playback. sudo apt install alsa-utils; aplay /usr/share/sounds/alsa/Front_Center.wav.

7.3 Conclusion

The 16GB "Atlas" workstation is a viable AI development platform only through rigorous subtraction. By removing the telemetry, prefetching, and container daemon overheads, we effectively "de-noise" the memory signal, allowing the AI workload to occupy the space previously held by the operating system's administrative bloat. The combination of **AtlasOS**, **Podman**, **Mirrored Networking**, and **Gradual Memory Reclamation** represents the state-of-the-art configuration for Windows 11 24H2 in January 2025.

Works cited

1. Windows Subsystem for Linux September 2023 update - Microsoft Dev Blogs, accessed on January 4, 2026, https://devblogs.microsoft.com/commandline/windows-subsystem-for-linux-september-2023-update/
2. Microsoft Rolling Out New Windows Subsystem For Linux "WSL" Features For 2024, accessed on January 4, 2026, https://www.phoronix.com/news/Microsoft-WSL2-2024-Features
3. High RAM Usage after Updating from Windows 11 23H2 to 24H2 - Microsoft Learn, accessed on January 4, 2026, https://learn.microsoft.com/en-us/answers/questions/5560338/high-ram-usage-after-updating-from-windows-11-23h2
4. High idle RAM usage across endpoints after upgrading to Windows 11 – normal or

leak?, accessed on January 4, 2026, https://www.reddit.com/r/sysadmin/comments/1ottko9/high_idle_ram_usage_across_endpoints_after/
5. Advanced settings configuration in WSL | Microsoft Learn, accessed on January 4, 2026, https://learn.microsoft.com/en-us/windows/wsl/wsl-config
6. AtlasOS - Optimized Windows, designed for enthusiasts., accessed on January 4, 2026, https://atlasos.net/
7. Supported Windows Versions - Atlas Documentation - AtlasOS, accessed on January 4, 2026, https://docs.atlasos.net/install-faq/windows-version-support/
8. Windows 11 25H2 : r/AtlasOS - Reddit, accessed on January 4, 2026, https://www.reddit.com/r/AtlasOS/comments/1o3pjtq/windows_11_25h2/
9. 5 Windows 11 services I disabled to give my PC a performance boost - Pocket-lint, accessed on January 4, 2026, https://www.pocket-lint.com/windows-services-i-disabled-for-performance-boost/
10. Should You Disable Sysmain in Windows? Boost Performance on Modern PCs, accessed on January 4, 2026, https://windowsforum.com/threads/should-you-disable-sysmain-in-windows-boost-performance-on-modern-pcs.376200/
11. How to Disable Superfetch (Sysmain) in Windows 11 - Acer Corner, accessed on January 4, 2026, https://blog.acer.com/en/discussion/2836/how-to-disable-superfetch-sysmain-in-windows-11
12. How Do I Fix High Memory Usage Windows 11? - AOMEI Partition Assistant, accessed on January 4, 2026, https://www.diskpart.com/windows-11/high-memory-usage-windows-11-0001.html
13. (NDU registry change) Increase performance immediately! AMAZING! - Page 22 - Hardware & Peripherals - Microsoft Flight Simulator Forums, accessed on January 4, 2026, https://forums.flightsimulator.com/t/ndu-registry-change-increase-performance-immediately-amazing/504551?page=22
14. Microsoft makes potential CPU, RAM, disk hogging feature default on Windows 11 25H2, 24H2 - Neowin, accessed on January 4, 2026, https://www.neowin.net/news/microsoft-makes-potential-cpu-ram-disk-hogging-feature-default-on-windows-11-25h2-24h2/
15. WSL is nice, but it takes up a lot of RAM on my system. I prefer to use MSYS2.... | Hacker News, accessed on January 4, 2026, https://news.ycombinator.com/item?id=39813522
16. WSL2 installation for Windows 10/11, accessed on January 4, 2026, https://pyyhttu.kapsi.fi/windows/wsl/wsl.html
17. Mirrored mode networking unavailable in Windows 11 Insider build 22635.4880 (and it should be) · Issue #12578 · microsoft/WSL - GitHub, accessed on January 4, 2026, https://github.com/microsoft/WSL/issues/12578
18. Mirrored Networking Mode stopped working with my WSL2 - Super User,

accessed on January 4, 2026, https://superuser.com/questions/1929757/mirrored-networking-mode-stopped-working-with-my-wsl2
19. CUDA on WSL User Guide - NVIDIA Documentation, accessed on January 4, 2026, https://docs.nvidia.com/cuda/wsl-user-guide/index.html
20. [Documented fix] Slow execution, Pytorch using GPU shared memory - windows, accessed on January 4, 2026, https://discuss.pytorch.org/t/documented-fix-slow-execution-pytorch-using-gpu-shared-memory/218909
21. PipeWire vs PulseAudio: What's the Difference? - It's FOSS, accessed on January 4, 2026, https://itsfoss.com/pipewire-vs-pulseaudio/
22. alsa vs pulseaudio vs jack vs pipewire : r/linuxaudio - Reddit, accessed on January 4, 2026, https://www.reddit.com/r/linuxaudio/comments/1jkvwb6/alsa_vs_pulseaudio_vs_jack_vs_pipewire/
23. USB microphone doesn't work using pipewire / Newbie Corner / Arch Linux Forums, accessed on January 4, 2026, https://bbs.archlinux.org/viewtopic.php?id=307066
24. Podman vs Docker Comparison: Performance, Security & Production [2025] - Uptrace, accessed on January 4, 2026, https://uptrace.dev/comparisons/podman-vs-docker
25. GPU container access - Podman Desktop, accessed on January 4, 2026, https://podman-desktop.io/docs/podman/gpu