

Decoding GPU Efficiency: Part 2 – A CTO's Dirty Dozen

February 20, 2026 · clockwork.io/blog/decoding-gpu-efficiency-part-2-a-ctos-dirty-dozen/

February 11, 2026

In [Part 1](#), we established the three-layer model for GPU Efficiency: GPU Allocation Utilization (GAU) × Effective Compute Time Ratio (ECTR) × Effective Model FLOPs Utilization (eMFU). We showed how a 75% GAU, 67% ECTR, and 40% eMFU compound into just 20% effective utilization.

For large-scale training, GAU losses claim 15–35%, ECTR losses claim a multiplicative 10–30%, and eMFU losses claim a multiplicative 50–70%. For production inference, the profile shifts: GAU can be dramatically better on elastic infrastructure or dramatically worse on always-on reserved instances; ECTR losses widen to 15–35% as prefill–decode interference and KV cache pressure replace data pipeline stalls; and eMFU losses widen to 55–75%, dominated by the fundamental memory-bandwidth bottleneck of autoregressive decoding.

This installment (Part 2) we name the twelve specific leaks responsible, ranked by impact. Four factors appear in both training and inference workloads, four are training-specific and four are inference-specific. Together, they are the Dirty Dozen.

The Dirty Dozen

The Four Leaks That Bleed Both Training and Inference Workloads

Leak 1: Memory-Bound Operations and Low Arithmetic Intensity

Training and Inference • eMFU layer

This is the single largest contributor to the gap between theoretical peak and achieved performance. It comes down to a fundamental imbalance in modern GPU architecture: the processors can perform math computations far faster than data can be delivered from High Bandwidth Memory (HBM) to the streaming multiprocessors. Operations with high arithmetic intensity, large matrix multiplications where data is reused many times, can saturate compute cores. Operations with low arithmetic intensity, such as element-wise additions, normalizations and attention on short sequences, spend most of their time shuffling data, not computing.

What to do: Obsess over arithmetic intensity. FlashAttention rewrites attention to operate within fast on-chip SRAM and minimize HBM round-trips. Kernel fusion combines sequential operations into a single data-load–compute–write cycle. Quantization (BF16 to FP8) halves bytes per element, effectively doubling arithmetic intensity. For inference, aggressive continuous batching converts the memory-bound matrix-vector decode operation back toward compute-bound matrix-matrix territory.

Leak 2: Idle Allocation and Over-Provisioning

Training and Inference • GAU layer

Organizations buy for peak but run at average. If peak demand is 100 GPUs and average demand is 40, you are paying for 60 idle GPUs during every off-peak period. In training, idle allocation is lumpy: a team reserves a 256-GPU cluster for a two-month training run, but the first week is spent on

debugging and data pipeline validation at a fraction of that scale, and the last few days are spent on evaluation and analysis that needs only a handful of GPUs. In inference, it is cyclical: a fleet provisioned for Tuesday afternoon peak is dramatically over-provisioned at 3am Sunday.

What to do: For training: right-size reservations to match run schedules, backfill idle periods with lower-priority jobs. For inference: adopt elastic autoscaling including scale-to-zero. For both, the decision between owning versus renting GPUs should be based on achievable utilization, not unit price alone: a \$2/hr reserved instance at 30% utilization costs more effectively than a \$4/hr serverless instance at 95%.

Leak 3: Whole GPU Allocation For Fractional Workloads

Training and Inference • GAU layer

Orchestrators assign entire GPUs to workloads that need a fraction of the device. A lightweight inference service needing 4 GB gets a device with 80 GB. You paid for an 8-lane highway and are using it for a single bicycle.

What to do: Implement GPU sharing through Multi-Instance GPU (MIG) partitioning, time-slicing via the NVIDIA GPU Operator, or commercial fractional schedulers like Run:ai.

Leak 4: Host-Side Overhead and Kernel Launch Latency

Training and Inference • ECTR layer

Every GPU operation must be launched by the CPU. Each launch costs tens of microseconds, and when you launch thousands of small kernels, this overhead dominates. For inference, where GPU computation per step can be very short, CPU launch overhead can be a significant fraction of total time.

What to do: CUDA Graphs capture entire operation sequences into a single replayable graph, eliminating per-kernel launch overhead. Batching inference requests aggressively where possible.

The Four Training Leaks That Hit Model Development

Leak 5: Failure Detection, Rollback Rework, and Cascade Preemptions

Training • eMFU/ECTR layer

At 1,024 GPUs, clusters experience approximately 3 disruptions per day. At 16,384 GPUs, that rises to over 12 per day. Each disruption forces a rollback to the last checkpoint, discarding all computation since. Writing a checkpoint for a large model can involve terabytes of data and pause computation across all GPUs. Each failure also incurs detection latency, re-provisioning cost, and can trigger cascade preemptions of lower-priority jobs.

What to do: Invest in observability for fast failure detection. Implement frequent asynchronous checkpointing, hot-spare nodes for instant replacement, and priority-aware scheduling to contain cascades. Modern, innovative approaches to fault-tolerant training that can migrate workloads non-disruptively can dramatically reduce the efficiency impact of disruptions.

Leak 6: Distributed Communication and Synchronization Overhead

Training • ECTR layer

As soon as training scales beyond a single GPU, communication eats into compute time. MegaScale reports a 6.2% MFU gain from improved communication–computation overlap alone. The overhead increases with scale and cannot be fully hidden behind computation on current hardware.

What to do: Engineer topology-aware placement, parallelism strategies that minimize cross-link data volume, and aggressive communication–computation overlap.

Leak 7: Pipeline Parallelism Bubbles

Training • eMFU layer

Pipeline parallelism divides a model into sequential stages across GPUs. The pipeline must fill and drain, creating “bubbles” where GPUs are allocated but idle. Think of a car wash with five stations: only when five cars are in the wash simultaneously does every station have work. The fill and drain time is pure waste.

What to do: Interleaved 1F1B (1 Forward, 1 Backward Pass) schedules, higher microbatch counts relative to pipeline stages, and zero-bubble or near-zero-bubble techniques like DeepSeek’s DualPipe.

Leak 8: The Activation Recomputation Trap

Training • eMFU layer

To save memory, engineers discard intermediate activations after the forward pass and recompute them during the backward pass. From the GPU’s perspective, this is real work. From the model’s perspective, it is the mathematical equivalent of driving in a circle. The odometer moves, fuel is burned, but you have not traveled anywhere. This creates a divergence between Hardware FLOPs (HFU) and Model FLOPs (MFU).

What to do: Selective activation checkpointing, memory-efficient attention (FlashAttention fuses recomputation into the backward pass at minimal cost), and offloading activations to CPU or NVMe.

The Four Inference Leaks That Hit Production Serving

Leak 9: KV Cache Memory Pressure and Fragmentation

Inference • ECTR/eMFU layer

Every token a transformer has processed leaves behind a key-value pair at every layer. For a 70B model with long context windows, a single request's KV cache can consume multiple gigabytes. Multiply by hundreds of concurrent requests, and **the KV cache, not the model weights, becomes the dominant consumer of GPU memory**. When memory fills, the system must reject requests, evict cached sequences, or recompute from scratch. Every outcome degrades effective utilization.

What to do: Implement paged KV cache management (vLLM's PagedAttention) to nearly eliminate fragmentation. Combine with prefix caching, KV cache compression, and intelligent eviction policies that offload least-recently-used pages to CPU memory rather than discarding them. For deployments with high context reuse, extend the cache hierarchy using GPUDirect Storage to offload KV cache to high-performance storage.

Leak 10: Prefill-Decode Phase Interference

Inference • ECTR layer

Inference serving is not one workload but two, awkwardly sharing the same GPU. Prefill processes the full input prompt in a single compute-bound forward pass. Decode generates tokens one at a time in a memory-bound trickle. When these run on the same GPU, they interfere – **like a sprinter and a marathon runner forced to share one lane**. Serving systems that try to balance both phases on a single GPU are forced into a compromise: either they let prefills interrupt decodes (harming latency for in-flight requests) or they delay prefills until decode batches drain (harming time-to-first-token (TTFT) for new requests).

What to do: Disaggregated serving separates prefill and decode onto different GPU pools. Chunked prefill offers a pragmatic middle ground for teams not ready for full disaggregation.

Leak 11: Request Batching and Scheduling Inefficiency

Inference • ECTR layer

Real-world inference traffic arrives one request at a time, at irregular intervals, with varying lengths. Static batching wastes GPU cycles in two ways: waiting to fill batches adds latency, and completed slots sit idle until the longest request finishes, like holding an entire restaurant table empty because one diner is still eating dessert. Continuous batching ejects finished requests immediately and inserts new arrivals at every decode step.

What to do: Make batching policy a product decision. Explicitly decide the queue delay vs. GPU efficiency tradeoff. Measure latency percentiles (p50, p95, p99) alongside utilization.

Leak 12: Speculative Decoding Opportunity Cost

Inference • eMFU layer

During autoregressive decode, the GPU has idle compute capacity at every step because the bottleneck is memory bandwidth, not compute. Speculative decoding uses a smaller draft model to propose multiple candidate tokens, then verifies them in a single batched pass through the full model. A draft model with 70–80% acceptance rate effectively multiplies decode throughput by 2–3×. **The opportunity cost of not using it is substantial.**

What to do: Integrate speculative decoding into the serving stack. Tune speculation depth (typically 3–7 tokens) based on workload acceptance rates. Leading frameworks (vLLM, TensorRT-LLM) now support this as a configurable option.

The Leaky Waterfall: Where 100 GPU-Hours Actually Go

The following tables map each leak to its impact layer, responsible team, and estimated hours lost from an initial 100 GPU-hours purchased.

Training: The Top 8 GPU Efficiency Leaks

Starting from 100 GPU-hours purchased, these are the largest sources of waste for large-scale model training.

Factor	Hours lost	Bucket	Team	Metric to obsess over
Memory Bound Operations and Low Arithmetic Intensity	10-13	eMFU	ML Research	Arithmetic Intensity (FLOPs/Byte)
Idle Allocation and Over-Provisioning	10-13	GAU	GPU Infra. Eng.	Unoccupied GPU-Hour Ratio
Failure Detection, Rollback Rework, Cascade Preemptions	9-10	eMFU/ ECTR	ML Platform / GPU Infra. Eng.	MTTD, MTTR, Job Goodput %
Whole-GPU Allocation for Fractional Workloads	8-9	GAU	GPU Infra. Eng.	Per-workload GPU Memory and Compute Utilization
Distributed Communication and Synchronization Overhead	5-7	ECTR	ML Research / GPU Infra. Eng.	Exposed Communication
Pipeline Parallelism Bubbles	5-7	eMFU	ML Research	Bubble Ratio
Activation Recomputation Trap	5-6	eMFU	ML Research	Gap between HFU and MFU
Host-Side Overhead & Kernel Launch Latency	2-4	ECTR	ML Research / ML Platform	Host-Bound Fraction

Inference: The Top 8 GPU Efficiency Leaks

Starting from 100 GPU-hours purchased, these are the largest sources of waste for production LLM inference serving.

Factor	Hours lost	Bucket	Team	Metric to obsess over
Decode Memory-Bandwidth Saturation	15-20	eMFU	ML Research / ML Platform	Arithmetic Intensity (FLOPs/Byte)
KV Cache Memory Pressure & Fragmentation	10-15	ECTR/ eMFU	ML Platform	KV Cache Hit Rate / Eviction Rate
Idle Allocation and Over-Provisioning	10-13	GAU	GPU Infra. Eng.	Unoccupied GPU-Hour Ratio
Prefill-Decode Phase Interference	8-12	ECTR	ML Platform / GPU Infra. Eng.	TTFT / Inter-Token Latency (ITL)
Whole-GPU Allocation for Fractional Workloads	6-8	GAU	GPU Infra. Eng.	Per-Workload GPU Memory and Compute Utilization
Request Batching and Scheduling Inefficiency	5-8	ECTR	ML Platform	Batch Utilization % / P95 Queue Delay
Host-Side Overhead & Kernel Launch Latency	4-6	ECTR	ML Research / ML Platform	Host-Bound Fraction
Speculative Decoding Opportunity Cost	2-4	eMFU	ML Research	Draft Acceptance Rate / Tokens per Forward Pass

The Bottom Line: The Dawn Of The Efficiency Era

For the last two years, the mandate for engineering leaders has been simple: **“Get GPUs at any cost.”**
The mandate for the next two years will be: **“Make every GPU-hour count.”**

What is clear is that simply relying on nvidia-smi utilization is the wrong metric. Start by measuring. If you don’t know your GAU, ECTR, and eMFU today, the most important thing you can do this quarter is instrument them.

The organizations that master this framework will not just save money. They will train faster, iterate more, and outcompete peers spending twice as much on the same hardware.

GPUs are expensive. Waste is *even more* expensive. The tools to measure and fix it exist.

What remains is the organizational discipline to use them.

References

1. nvidia-smi queries NVIDIA’s NVML (NVIDIA Management Library), which samples at roughly 1-second intervals and reports what percentage of that interval had at least one kernel active on the GPU. This is essentially a binary occupancy check: during the sample window, was something running on the GPU or not? A kernel that lights up 5% of the streaming multiprocessors counts the same as one saturating every core at peak throughput.
2. ‘I paid for the whole GPU, I am going to use the whole GPU’: A high-level guide to GPU utilization, Modal Blog
3. Revisiting Reliability in Large-Scale Machine Learning Research Clusters, arXiv:2410.21680

4. Building Meta's GenAI Infrastructure, Meta Engineering Blog
5. The Llama 3 Herd of Models, arXiv:2407.21783
6. <https://docs.aws.amazon.com/eks/latest/best-practices/aiml-observability.html>
7. FlashRecovery: Fast and Low-Cost Recovery from Failures for Large-Scale Training of LLMs
arXiv:2509.03047v1
8. Amazon Science Blog, "More Efficient Recovery from Failures During Large ML Model Training."
9. MegaScale: Scaling Model Training to More Than 10,000 GPUs (NSDI 2024)
10. MegaScale-MoE: Large-Scale Communication-Efficient Training of Mixture-of-Experts Models in Production," arXiv:2505.11432, May 2025
11. "Reducing Activation Recomputation in Large Transformer Models", arXiv:2205.05198
12. PaLM: Scaling Language Modeling with Pathways, arXiv:2204.02311
13. Performance Deep Dive of Gemma on Google Cloud," Google Cloud Blog, April 2024.
14. The State of AI Infrastructure at Scale 2024," ClearML, AI Infrastructure Alliance, and FuriosaAI, March 2024
15. DeepSeek-V3 Technical Report," arXiv:2412.19437, December 2024.
16. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness,"
arXiv:2205.14135, May 2022.
17. Improving GPU Utilization in Kubernetes: <https://developer.nvidia.com/blog/improving-gpu-utilization-in-kubernetes/>
18. Efficient Memory Management for Large Language Model Serving with Paged Attention," SOSP 2023