GPU SMITH
COMPUTE ENGINEERING

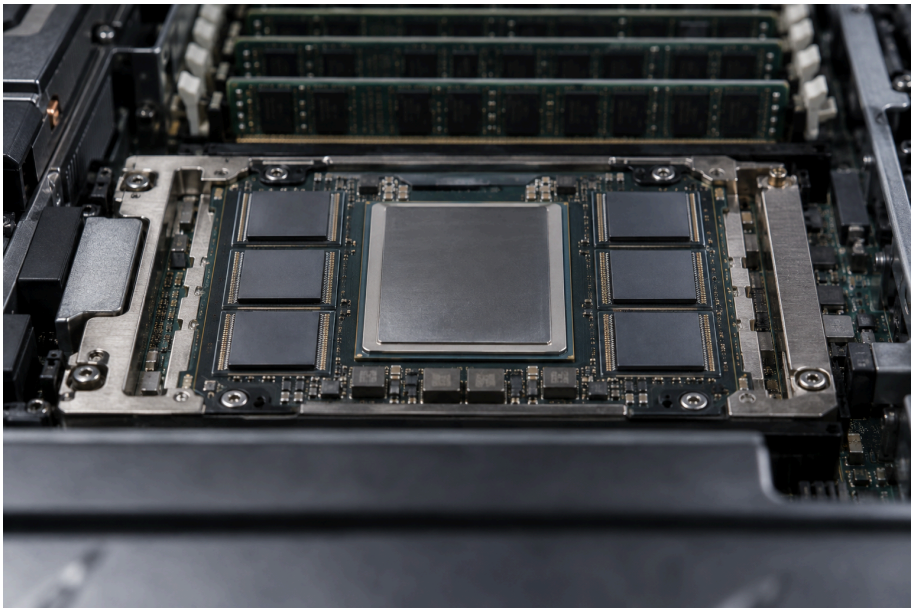
GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

LLM Training Memory Calculator for ZeRO and FSDP

> Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-09-19 · llm training memory calculator · zero memory calculator · fsdp memory calculator · gpu memory · distributed training · deepspeed · pytorch fsdp · cpu offload · gpu sizing

Original article: https://gpusmith.com/articles/en/llm-training-memory-calculator



In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes
4. Full Training, LoRA, QLoRA, and Offload Branches
5. Implementation Considerations and Process Changes
6. Data Analysis and Evidence
7. Implications and Future Directions
8. Frequently Asked Questions (FAQs)
9. Conclusion

Executive Summary

An **LLM training memory calculator** should produce a per-rank range, not a binary promise. The lower bound is the sum of persistent model states divided only where the selected strategy actually shards them. The upper planning bound adds activations, the largest simultaneously gathered parameter unit, communication and optimizer temporaries, runtime allocations, and explicit headroom. This distinction matters because DeepSpeed's own state estimator says activation and intermediate memory must be added separately (deepspeed.readthedocs.io), while PyTorch exposes allocated and allocator-reserved memory as different quantities (docs.pytorch.org). A procurement decision should therefore carry a versioned estimate, sensitivity range, and measured validation record.

For a common mixed-precision Adam configuration, one documented ledger is **6 bytes per parameter for FP16 weights plus an FP32 main copy**, **4 bytes for FP32 gradients**, and **8 bytes for two FP32 Adam states**, or 18 bytes before activations and transients (huggingface.co) (huggingface.co). That is a configuration, not a universal rule. FSDP2, for example, can retain high-precision sharded parameters without another high-precision optimizer copy (docs.pytorch.org). The calculator must ask for the exact optimizer, precision policy, framework version, and implementation.

ZeRO stage 1 shards optimizer state, **stage 2** also shards gradients, and **stage 3** also partitions parameters and gathers them around computation (deepspeed.ai) (deepspeed.ai). PyTorch **FSDP FULL_SHARD** likewise shards parameters, gradients, and optimizer states (docs.pytorch.org). Yet average state divided by rank count is not peak HBM: an AllGather can materialize one or more layers in transient output buffers (aws.amazon.com).

The resulting distributed LLM training GPU calculator should report three counts: the mathematical minimum, the topology-compatible count, and a validation range. Megatron Core expresses total ranks as **TP x PP x CP x EP x DP** (docs.nvidia.com). Capacity should be shown in both decimal GB and binary GiB, where **1 GiB is 1,073,741,824 bytes** and **1 GB is 1,000,000,000 bytes** (physics.nist.gov) (physics.nist.gov). Offload can change the capacity ledger, but it cannot establish acceptable throughput. Final approval requires a representative microbatch on the pinned stack, with peak allocated, peak reserved, non-framework allocations, and checkpoint staging observed.

Introduction and Background

The planning question is deceptively simple: can a proposed full-training or fine-tuning job fit, and how many accelerators does it require? A correct answer spans four coupled ledgers: GPU high-bandwidth memory (HBM), host random-access memory (RAM), local or shared non-volatile memory express (NVMe) storage, and checkpoint storage. It also spans time. Parameters may be sharded while idle, gathered for a layer, prefetched for the next layer, and consolidated again for a checkpoint.

The report uses **per rank** to mean the memory visible to one training process, ordinarily one accelerator. It separates persistent state from peak additions, and it treats all user inputs as part of a versioned record. This is necessary because ordinary data parallelism gives every GPU a complete model copy (docs.nvidia.com),

while tensor parallelism (TP) splits selected layers and pipeline parallelism (PP) assigns different layers to stages (docs.nvidia.com) (docs.nvidia.com). Neither operation is equivalent to dividing every byte by the total GPU count.

As of **September 19, 2026**, the framework and hardware version are first-class inputs. Current PyTorch documentation distinguishes FSDP1 from per-parameter FSDP2, and the migration tutorial maps FULL_SHARD to `reshard_after_forward=True` behavior (docs.pytorch.org). Hardware capacity also changes the topology choice: an eight-GPU DGX H100 has **640 GB** aggregate GPU memory, while an eight-GPU DGX H200 has **1,128 GB** (docs.nvidia.com) (docs.nvidia.com). Aggregate capacity is descriptive, however. Fit must hold on every rank, including the most heavily loaded pipeline stage.

GPU Smith is an adjacent infrastructure advisor, not a competing training framework. Its published process starts with workload modelling and a reference architecture and ends with documented acceptance testing (gpusmith.com) (gpusmith.com). That posture is useful here: the calculator is an auditable requirements model, not a substitute for a measured acceptance test.

Key Changes

Change 1: Replace one bytes-per-parameter constant with a ledger

A planning model starts with named tensors, not folklore. Let `P` be total parameters and `T` be trainable parameters. For every state `i`, record count `N_i`, storage bytes `b_i`, replication factor `r_i`, sharding divisor `s_i`, and location. The persistent bytes on rank `j` are the sum of `N_i x b_i x r_i / s_i`, adjusted for stage placement and imbalance. This notation exposes whether a value applies to `P`, `T`, a local pipeline partition, or a selected tensor-parallel shard.

Table 1 defines the minimum input and output ledger for an enterprise calculator.

Ledger row	Required inputs	Per-rank device calculation	What must remain explicit
Weights and main weights	Parameter count, storage dtype, compute dtype, master-weight policy	Count x actual bytes x replica factor / applicable shard degree	FP16 is 16 bits and FP32 is 32 bits (docs.nvidia.com); a master copy is optional and implementation-specific.
Gradients	Trainable count, reduction dtype, accumulation policy	`T x gradient bytes / gradient shard degree`	Full training uses `T=P`; frozen-base adaptation does not.
Optimizer	Named optimizer, state count and dtype, implementation	Sum of each state divided by its optimizer shard degree	Documented Adam momentum plus variance can add 8 bytes per parameter (deepspeed.readthedocs.io).
Activations	Architecture, layers, hidden size, attention type, sequence length, packing, microbatch, checkpointing	Architecture formula or calibrated range for the local stage	Activation size varies with batch, sequence, depth, and hidden size (huggingface.co).

Ledger row	Required inputs	Per-rank device calculation	What must remain explicit
Peak temporaries	Largest wrapped unit, collective bucket, prefetch depth, optimizer kernel	Measured or documented upper range, not amortized	PyTorch foreach AdamW uses roughly one parameter set more peak memory than the for-loop form (docs.pytorch.org).
Runtime and safety	CUDA context, libraries, allocator reserve, uncaptured allocations, explicit margin	Fixed measured base plus percentage or absolute reserve	Cached unused memory can still appear used in `nvidia-smi` (docs.pytorch.org).
Host, NVMe, checkpoint	Offloaded rows, staging copies, shard format, retention count	Separate capacity totals, never subtracted without relocation	Full-state consolidation can require substantial CPU memory (deepspeed.readthedocs.io).

The ledger prevents two frequent category errors. First, an FP32 master copy should not be counted because a generic article once counted it. NVIDIA describes master weights as one solution for optimizer updates (docs.nvidia.com), while its performance guidance says different arrangements have different footprints (docs.nvidia.com). Second, a decimal device label must not be mixed with a binary calculation. The output should show raw bytes, GB, and GiB side by side.

Change 2: Model average shards and peak gathers separately

The **ZeRO memory calculator** branch needs a strategy table plus a peak-event model. Stage 1 partitions optimizer states. Stage 2 additionally partitions gradients. Stage 3 partitions the 16-bit parameters and gathers them as computation requires (deepspeed.ai). DeepSpeed's published lower-bound expression for stage 3 without offload is largest-layer memory plus `18 x P / total GPUs` for the documented configuration (deepspeed.readthedocs.io). The important term is the largest layer: it survives even when persistent state is deeply sharded.

Table 2 maps the major strategies. “Peak addition” is deliberately qualitative because wrapper boundaries, prefetch policy, buckets, and release timing determine its size.

Strategy	Parameters at rest	Gradients	Optimizer states	Peak addition to model
DDP or ZeRO 0	Replicated	Replicated	Replicated	Collective and optimizer temporaries; no state-sharding relief.
ZeRO 1	Replicated	Replicated	Sharded over DP group	Weight and gradient replication remains.
ZeRO 2	Replicated	Sharded over DP group	Sharded over DP group	Parameters remain resident; reduction buckets still peak.
ZeRO 3	Sharded over DP group	Sharded over DP group	Sharded over DP group	Largest gathered unit, prefetch bucket, current gradients, and other live units. DeepSpeed says enough HBM is needed to gather the largest layer on one GPU (deepspeed.readthedocs.io).

Strategy	Parameters at rest	Gradients	Optimizer states	Peak addition to model
FSDP1 SHARD_GRAD_OP	Parameters sharded outside computation	Sharded	Sharded	Parameters can remain unsharded through forward and backward; it is stage-2-like (docs.pytorch.org).
FSDP1 FULL_SHARD	Sharded	Sharded	Sharded	Current unit plus prefetched unit can coexist. BACKWARD_PRE may hold current parameters, next parameters, and current gradients (docs.pytorch.org).
FSDP2, reshard after forward	Sharded outside compute	Sharded	Sharded	Resharding lowers retained parameters but requires another backward all-gather (docs.pytorch.org).
Hybrid sharding	Sharded within a group, replicated across groups	Same group policy	Same group policy	One full model-state copy per replica group; Hugging Face describes within-node sharding plus cross-node replication (huggingface.co).

The table answers **ZeRO stage 3 memory requirements** and **FSDP GPU memory requirements** at the right level: persistent states divide, but live unsharded units do not. FSDP's all-gather limiter is intended to constrain memory to two consecutive FSDP instances (docs.pytorch.org). DeepSpeed similarly exposes a prefetch bucket measured as the maximum number of elements fetched ahead (deepspeed.readthedocs.io). Those configuration values belong in the calculator schema.

Peak control is also an implementation choice. DeepSpeed documents sequential all-gather as a way to avoid large temporary flattening buffers (deepspeed.readthedocs.io). Red Hat's communication description confirms that sharded weights may be gathered on every GPU before each layer's forward and backward passes (developers.redhat.com). These controls should be inputs, never hidden constants.

Change 3: Treat parallelism as a matrix, not one divisor

The **distributed LLM training GPU calculator** should calculate a world-size lattice before it calculates a purchase quantity. The main dimensions are:

- **Data parallelism (DP):** replicas process different samples; the ZeRO or FSDP sharding group is commonly tied to this dimension.
- **Tensor parallelism (TP):** selected matrices and their work are divided within a layer. Parameters outside those plans may remain replicated.
- **Pipeline parallelism (PP):** layer groups are assigned to stages. The maximum stage, not the average stage, determines fit.
- **Context parallelism (CP):** sequence work is partitioned where the framework supports it; it does not imply optimizer-state division.
- **Expert parallelism (EP):** mixture-of-experts modules distribute experts, while shared dense tensors remain subject to their own plan. Expert parallelism also introduces token dispatch, for which Megatron recommends an all-to-all dispatcher (docs.nvidia.com).

- **Sequence parallelism and replication:** these are additional flags on selected tensors, not automatic divisors for the whole ledger.

The valid total is `TP x PP x CP x EP x DP`, subject to framework compatibility. The sharding divisor for model states is the group that actually shards them, not necessarily world size; AWS describes state partitioning across GPUs within a data-parallel group (docs.aws.amazon.com). In a layout illustration (Hypothetical Example), a 64-GPU job with TP=8, PP=2, and DP=4 does not automatically give a 64-way optimizer-state reduction. If sharding is scoped to DP, the divisor is 4.

Topology changes the rounded answer. AWS recommends power-of-two TP degrees (docs.aws.amazon.com) and, for multi-node TP plus PP, recommends keeping TP within nodes and placing PP across nodes (docs.aws.amazon.com). In a rounding illustration (Hypothetical Example), the arithmetic minimum may be 10 GPUs while the first legal or operationally sensible layout is 16. The output must retain both numbers and the rounding reason.

Full Training, LoRA, QLoRA, and Offload Branches

Full training and parameter-efficient adaptation

Full fine-tuning retrains every model parameter (arxiv.org), so `T=P` for gradients and optimizer states. Low-Rank Adaptation (LoRA) constrains a dense update to a low-rank decomposition rather than learning the complete update matrix (arxiv.org). In the Hugging Face Parameter-Efficient Fine-Tuning (PEFT) implementation, only adapter parameters are updated, greatly narrowing optimizer and gradient scope (huggingface.co).

The calculator must use separate branches:

- **Full training:** base weights, gradients, optimizer states, and any main weights apply across all trainable parameters.
- **LoRA:** base weights stay resident according to their storage policy, while adapter parameters create the trainable gradient and optimizer ledger.
- **QLoRA:** the frozen base is stored in 4-bit form and gradients pass through it into trainable LoRA adapters (arxiv.org). The base is dequantized to BFloat16 for matrix multiplication (arxiv.org), so storage bits alone do not describe peak compute buffers.
- **Custom partial tuning:** the user supplies a trainable mask or count. No frozen-base assumption is inferred from a method label.

QLoRA's paper reports **0.373 bits per parameter** less metadata under its double-quantization block assumptions (arxiv.org). That is useful for reproducing that configuration, but it is not a universal 4-bit overhead. More importantly, the paper says activation gradients account for most LLM fine-tuning memory (arxiv.org). Reducing trainable state does not remove the activation branch.

The implementation must agree with the method label. Hugging Face states that training with 8-bit or 4-bit base weights supports updating extra parameters rather than those base weights (huggingface.co). Its LoRA reference also distinguishes ordinary projection targeting from QLoRA-style adapters on every linear transformer layer (huggingface.co). Adapter target lists, ranks, and dtypes therefore belong in the serialized input.

CPU and NVMe offload

An **LLM CPU offload calculator** is three linked capacity ledgers plus a transfer path. It should expose:

- **Device-resident bytes:** shards, live gathered parameters, activation range, current collectives, and runtime reserve.
- **Host-resident bytes:** offloaded parameters, gradients or optimizer states, pinned transfer buffers, checkpoint consolidation, dataloader demand, and operating-system reserve.
- **NVMe working set:** offloaded state, queue or staging buffers, checkpoint shards, temporary full-state materialization, and retention policy.
- **Transfer path:** device to host, host to NVMe, peer collectives, and whether computation moves with the state.
- **Performance constraints:** buyer-supplied minimum samples per second, maximum step time, and acceptable checkpoint window, validated rather than inferred from capacity.

DeepSpeed parameter offload is valid only with stage 3 (deepspeed.readthedocs.io). Its optimizer computation runs on CPU even when NVMe is the backing device (deepspeed.readthedocs.io). FSDP2's CPU policy moves parameters, gradients, and optimizer states to CPU and consequently places the optimizer step there (docs.pytorch.org). These facts describe placement, not throughput. A “fits in RAM” result must never be translated into a training-rate claim.

Implementation Considerations and Process Changes

Versioned input schema

Every run should serialize enough information to reproduce the estimate:

- **Identity:** calculator schema version, timestamp, model revision, architecture, parameter count, largest wrapped unit, and mixture-of-experts structure.
- **Training method:** pretraining, full fine-tuning, LoRA, QLoRA, or a custom trainable subset.
- **Software:** framework, DeepSpeed or PyTorch version, optimizer class and implementation, attention kernel, precision policy, compiler settings, and allocator configuration.
- **Batch geometry:** sequence length, packing distribution, microbatch per rank, gradient accumulation, and global batch.

- **Memory controls:** activation checkpoint policy, CPU or NVMe offload, collective bucket sizes, prefetch policy, wrapping boundaries, and reshard timing.
- **Parallel dimensions:** DP, TP, PP, CP, EP, sharding group, replication group, and rank-to-node placement.
- **Hardware:** usable HBM per rank, host RAM per node, NVMe free space, accelerators per node, fabric, and reserved capacity policy.
- **Output policy:** arithmetic fit margin, topology rounding, required confidence band, checkpoint retention, and acceptance thresholds.

Activation checkpointing is not a simple percentage toggle. It reduces saved tensors by recomputing them during backward (pytorch.org). Standard self-attention is quadratic in sequence length for both time and memory (arxiv.org), while FlashAttention avoids storing the large intermediate attention matrix for backward (arxiv.org). Architecture, kernel, sequence packing, and checkpoint policy must therefore select or calibrate the activation formula.

GPU-count output and validation plan

The **LLM training GPU count calculator** should emit these outputs in order:

1. **Persistent per-rank state:** lower-bound arithmetic for the selected sharding group.
2. **Peak event additions:** activations, largest gathered unit, prefetch, reductions, optimizer temporaries, and runtime range.
3. **Raw fit count:** the smallest rank count whose high estimate is below usable HBM.
4. **Legal parallel layout:** the first TP x PP x CP x EP x DP product that satisfies divisibility and placement constraints.
5. **Node-rounded procurement count:** complete nodes or the explicitly supported partial-node shape.
6. **Host and storage requirement:** per node and aggregate, including checkpoint staging.
7. **Validation range:** at least the topology-rounded minimum and the next practical layout when uncertainty overlaps the HBM limit.
8. **Reasons:** dominant memory rows, unverified assumptions, and which measurement can narrow the range.

Validation begins before the whole model is allocated. FSDP2 documentation supports initialization after sharding, avoiding a full pre-sharding model allocation (docs.pytorch.org). Then run a representative microbatch and record:

- **Configuration hash:** code, model revision, exact command, environment, and serialized calculator input.
- **Peak allocated HBM:** the maximum held by live tensors.

- **Peak reserved HBM:** the caching allocator's maximum managed amount.
- **External device memory:** libraries such as NCCL may allocate memory outside PyTorch's profiler visibility (docs.pytorch.org).
- **Fragmentation evidence:** memory snapshot, segment distribution, and any allocator changes. PyTorch documents split-size tuning as a possible aid for borderline fragmentation cases (docs.pytorch.org).
- **Steady-state window:** exclude warm-up allocations. TensorFlow Profiler likewise advises avoiding the first batches because initialization can distort results (tensorflow.org).
- **Error decomposition:** measured peak minus estimated midpoint, attributed to activations, collective overlap, optimizer temporaries, runtime, or untracked allocations.
- **Checkpoint test:** save, load, reshard, and stage a checkpoint under the intended topology. PyTorch Distributed Checkpoint can load-time reshard between cluster topologies (docs.pytorch.org).

Data Analysis and Evidence

Worked planning calculation (Hypothetical Example)

Consider a fictional **7 billion parameter** dense model under full fine-tuning. This is a unit-checking example, not a benchmark. Assume the documented 18-byte mixed-precision Adam ledger, an eight-rank DP sharding group, a user-measured **0.6 GB** largest wrapped unit, **12 to 20 GiB** of activations, **2 to 4 GiB** of collective and optimizer temporaries, **3 to 5 GiB** of runtime allocations, and **15 percent** safety headroom. No throughput is claimed.

The unsharded state is `7,000,000,000 x 18 = 126,000,000,000 bytes`, or **126 GB and 117.35 GiB**. Stage 3 or full sharding divides that persistent state by eight to **15.75 GB, or 14.67 GiB**, then adds the **0.56 GiB** largest unit. Before headroom, the range is `14.67 + 0.56 + 12 + 2 + 3 = 32.23 GiB` to `14.67 + 0.56 + 20 + 4 + 5 = 44.23 GiB`. Applying 15 percent headroom yields **37.07 to 50.86 GiB per rank**.

Table 3 compares outputs from the same declared assumptions.

Output	ZeRO 3 or FSDP full-shard branch	ZeRO 2 or stage-2-like branch	Decision use
Persistent model state	14.67 GiB sharded state plus 0.56 GiB gathered unit	39.12 GiB replicated 6-byte weight/main-weight set, plus 3.26 GiB sharded gradients and 6.52 GiB sharded Adam states	Shows why stage selection changes HBM rather than merely communication.
Planned per-rank range	**37.07 to 50.86 GiB** after 15 percent headroom	**76.43 to 90.23 GiB** after the same additions and headroom	Both are assumption-driven capacity ranges, not measured peaks.
Eight-rank device interpretation	Fits arithmetically on an 80 GB-class rank under stated assumptions	Fits arithmetically, but with materially less remaining capacity	Proceed to measured validation on the pinned stack.

Output	ZeRO 3 or FSDP full-shard branch	ZeRO 2 or stage-2-like branch	Decision use
Host and NVMe result	Not calculated until specific rows are offloaded and staging policy is declared	Same	Capacity is a separate ledger; no speed conclusion follows.
Procurement output	Arithmetic minimum: 8; topology-compatible count: 8; validation range: 8 to 16	Arithmetic minimum: 8 under these inputs; validation range may expand if measured peak exceeds estimate	Round by complete supported node and parallel layout, not fractional aggregate HBM.

This comparison isolates the consequence of parameter replication. It does not prove that stage 2 or stage 3 is faster, because communication overlap, kernel choice, interconnect, and offload paths are outside a capacity-only calculation. Pipeline schedules also change activation peaks: every concurrently active microbatch carries activations and gradients (docs.aws.amazon.com).

Hardware capacity is necessary but not sufficient

Official system specifications illustrate why node layout belongs beside per-rank output. An eight-GPU DGX B200 supplies **1,440 GB** aggregate GPU memory (docs.nvidia.com). An eight-accelerator AMD MI300X baseboard supplies **1.5 TB HBM3**, or **192 GB per accelerator** (instinct.docs.amd.com) (instinct.docs.amd.com). DGX H100 and H200 list **2 TB** of host memory (docs.nvidia.com).

The physical fabrics differ as well. DGX H100 and H200 document **900 GB/s GPU-to-GPU bandwidth** through fourth-generation NVLink (docs.nvidia.com), while DGX B200 documents **14.4 TB/s aggregate** fifth-generation NVLink switch bandwidth (docs.nvidia.com). AMD describes the eight-accelerator MI300X platform as fully meshed over Infinity Fabric (instinct.docs.amd.com). These are topology inputs, not interchangeable throughput predictions.

Those specifications constrain the ledgers, but they do not replace them. Aggregate HBM cannot repair one oversized pipeline stage. Host capacity cannot be allocated entirely to optimizer state because the operating system, data pipeline, pinned buffers, and checkpoint consolidation also need space. Interconnect bandwidth does not establish end-to-end training throughput without message sizes and overlap. The calculator should preserve every capacity and performance constraint as a separately testable claim.

Offload expands the usable hierarchy rather than erasing movement costs. The ZeRO-Infinity paper explicitly spans GPU, CPU, and NVMe memory (arxiv.org). Likewise, an official server specification can confirm an eight-GPU H100 configuration (docs.nvidia.com), but only a workload trace can show whether the planned traffic meets the acceptance threshold.

Implications and Future Directions

For enterprise platform and finance teams, the most important implication is that **GPU memory requirements for LLM training** are not one scalar. They are a versioned distribution over ranks and training phases. A defensible procurement packet should include the input schema, formula version, low and high estimate by ledger row, topology diagram, configuration hash, validation logs, and the exception list.

Several process changes follow:

- **Procure against a validated topology:** buy complete supported units after TP, PP, and sharding groups are laid out.
- **Retain uncertainty explicitly:** use a range until activation and temporary allocations are measured.
- **Separate capacity from performance:** offload can make a state ledger fit while failing a buyer's step-time threshold.
- **Treat checkpointing as a peak event:** include save, load, consolidation, and topology change in host and storage capacity.
- **Recalculate after software changes:** framework releases, optimizer implementations, wrap policies, kernels, and allocator settings can move the peak.
- **Test the heaviest rank:** pipeline imbalance and expert placement can make the average misleading.
- **Record both allocated and reserved memory:** a single dashboard number cannot explain allocator headroom.

Future calculator versions should add architecture-specific activation plugins for grouped-query attention, mixture-of-experts routing, sequence packing distributions, and fused optimizers. They should also ingest a measured pilot trace and update uncertain rows instead of applying an opaque correction factor. This creates a useful feedback loop: arithmetic proposes the topology, a small-scale or full-rank test measures the peak, and the next version reconciles the difference.

Frequently Asked Questions (FAQs)

How many GPUs are needed to train an LLM?

Calculate per-rank high memory under each legal parallel layout, select the smallest layout below usable HBM, then round to a supported node topology. Report the raw minimum, topology-rounded count, and validation range. Do not divide total bytes by aggregate HBM alone.

What does a ZeRO memory calculator need beyond parameter count?

It needs optimizer states, gradient and parameter dtypes, main-weight policy, trainable count, DP sharding degree, largest gathered unit, collective buckets, prefetch, activations, runtime reserve, offload placement, and headroom. DeepSpeed's cold estimator can avoid loading the model after parameter counts are known (deepspeed.readthedocs.io), but its state estimate still does not replace the activation and temporary ledger.

What should an FSDP memory calculator report?

It should report persistent shards by FSDP strategy, the largest unsharded unit, prefetch overlap, reshard timing, activation and runtime ranges, host placement, and both allocated and reserved measured peaks. The calculation must also name whether it targets FSDP1 or FSDP2 because their state representation and controls differ.

Are ZeRO stage 3 and FSDP FULL_SHARD identical?

They share the broad policy of sharding parameters, gradients, and optimizer states. PyTorch explicitly calls FSDP its ZeRO-3 implementation (pytorch.org). They are not interchangeable calculator profiles because gather timing, wrapping, prefetch, optimizer representation, checkpoint APIs, and versions differ.

Does CPU or NVMe offload guarantee a job will run fast enough?

No. It can solve a device-capacity constraint by relocating state, but the result still depends on transfer volume, overlap, CPU optimizer work, storage behavior, and the workload's required step time. Capacity fit and throughput acceptance are separate gates.

Why can measured memory exceed the estimate?

Likely causes include activation assumptions, overlapping gathered units, optimizer temporaries, allocator reserve, fragmentation, CUDA context and libraries, and allocations outside the framework profiler. Short-lived temporaries can create an out-of-memory spike even when steady state appears safe (huggingface.co).

Conclusion

A useful LLM training memory calculator is a transparent model of tensors, placement, time, and topology. It starts with a byte-accurate persistent-state ledger, applies only the sharding divisors that the pinned framework actually uses, and adds the peak events that average arithmetic hides. It handles full training, LoRA, and QLoRA as different branches. It reports GPU, host, NVMe, and checkpoint capacity independently.

The decision output is a range with three GPU counts: mathematical minimum, topology-compatible procurement count, and validation range. A result is ready for procurement only after a representative microbatch and checkpoint path have been measured on the intended software stack, with allocated, reserved, and external device memory recorded. That approach turns a convenient estimate into an auditable infrastructure requirement without pretending that capacity alone predicts training performance.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://deepspeed.readthedocs.io/en/latest/memory.html
2. https://docs.pytorch.org/docs/main/notes/cuda.html
3. https://huggingface.co/docs/transformers/model_memory_anatomy
4. https://docs.pytorch.org/docs/main/distributed.fsdp.fully_shard.html
5. https://www.deepspeed.ai/tutorials/zero/
6. https://docs.pytorch.org/docs/main/fsdp.html
7. https://aws.amazon.com/blogs/machine-learning/enable-faster-training-with-amazon-sagemaker-data-parallel-library/
8. https://docs.nvidia.com/megatron-core/developer-guide/latest/user-guide/parallelism-guide.html
9. https://physics.nist.gov/cuu/Units/binary.html
10. https://docs.pytorch.org/tutorials/intermediate/FSDP_tutorial.html

11. https://docs.nvidia.com/dgx/dgxh100-user-guide/introduction-to-dgxh100.html
12. https://gpusmith.com/
13. https://gpusmith.com/articles/en/gpu-cluster-acceptance-testing-runbook
14. https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html
15. https://docs.pytorch.org/docs/main/generated/torch.optim.AdamW.html
16. https://deepspeed.readthedocs.io/en/latest/model-checkpointing.html?highlight=deepspeed
17. https://docs.nvidia.com/deeplearning/transformer-engine-releases/release-2.15/user-guide/features/low_precision_training/introduction/introduction.html
18. https://docs.nvidia.com/deeplearning/transformer-engine-releases/release-2.14/user-guide/features/low_precision_training/performance_considerations/performance_considerations.html
19. https://huggingface.co/docs/accelerate/main/usage_guides/fsdp
20. https://deepspeed.readthedocs.io/en/latest/_modules/deepspeed/runtime/zero/config.html
21. https://developers.redhat.com/articles/2025/04/29/accelerate-model-training-openshift-ai-nvidia-gpudirect-rdma
22. https://docs.nvidia.com/megatron-core/developer-guide/0.15.0/api-guide/moe.html
23. https://docs.aws.amazon.com/sagemaker/latest/dg/model-parallel-intro.html
24. https://docs.aws.amazon.com/sagemaker/latest/dg/model-parallel-best-practices.html
25. https://arxiv.org/html/2106.09685
26. https://huggingface.co/docs/transformers/main/peft
27. https://arxiv.org/html/2305.14314
28. https://huggingface.co/docs/transformers/v4.48.0/quantization/bitsandbytes
29. https://huggingface.co/docs/peft/main/package_reference/lora
30. https://deepspeed.readthedocs.io/en/stable/_modules/deepspeed/runtime/zero/offload_config.html
31. https://pytorch.org/blog/activation-checkpointing-techniques/
32. https://arxiv.org/abs/2205.14135
33. https://docs.pytorch.org/docs/main/torch_cuda_memory
34. https://www.tensorflow.org/guide/profiler
35. https://docs.pytorch.org/docs/main/distributed.checkpoint.html
36. https://docs.nvidia.com/dgx/dgxb200-user-guide/introduction-to-dgxb200.html
37. https://instinct.docs.amd.com/projects/system-acceptance/en/latest/gpus/mi300x.html
38. https://gpusmith.com/articles/en/nvlink-5-vs-nvlink-6
39. https://arxiv.org/abs/2104.07857
40. https://pytorch.org/blog/training-moes/

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the hardware library, GPU references, networking references and vendor research. Contact GPU Smith with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: https://gpusmith.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.