Qwen

A Practical Guide to Hyperparameter Configuration for ConvNeXt Training on NVIDIA RTX 4090 Hardware

Architectural Foundations: The Impact of RTX 4090 Constraints on Training Strategy

A deep understanding of the architectural underpinnings of the NVIDIA GeForce RTX 4090 is not merely an academic exercise; it is the foundational prerequisite for any successful and efficient training strategy. Launched in late 2022, the RTX 4090 represents a pinnacle of consumer-grade GPU technology, built on the TSMC 4N process and featuring the AD102 GPU die[35]. Its specifications—24 GB of high-speed GDDR6X VRAM, 1 TB/s of memory bandwidth, 16,384 CUDA cores, and 512 fourth-generation Tensor Cores—position it as a formidable tool for deep learning workloads, including the training of complex convolutional networks like ConvNeXt[24][27][32]. However, its true power and limitations are defined by a critical omission from its design: the lack of support for NVLink, peer-to-peer (P2P) communication, and SLI[9][25][31]. This single characteristic profoundly shapes the landscape of both single-GPU and multi-GPU training, necessitating a strategic approach to hyperparameter selection and parallelism that diverges significantly from enterprise-class server GPUs equipped with high-bandwidth interconnects like NVLink or InfiniBand[57].

For a single RTX 4090 setup, the most immediate and constraining factor is its 24 GB VRAM capacity[3][28]. This finite resource dictates the maximum feasible batch size and model complexity without the application of aggressive memory-saving techniques. The total VRAM footprint for a training session is a composite of several components: the model's own parameters, the gradients computed during backpropagation, the optimizer states (such as those for AdamW, which consume approximately 12 bytes per parameter), and the intermediate activation tensors stored for the forward and backward passes[15][26]. For instance, a full fine-tune of a 7-billion-parameter model in 16-bit precision requires roughly 112 GB of VRAM (16 bytes per parameter × 7 billion parameters), rendering it entirely infeasible on a

single card [23] [69]. This hard constraint elevates quantization-based Parameter-Efficient Fine-Tuning (PEFT) methods, particularly QLoRA, from a niche optimization to a primary enabling technology for training models larger than ~7 billion parameters [36] [71]. Furthermore, the Ada Lovelace architecture incorporates significant architectural improvements over its predecessors, such as a massive 72 MB L2 cache, which reduces memory latency bottlenecks and improves performance for models where weight matrices fit within this cache, yielding up to a 2.0x speedup in some cases [29] [33]. The fourth-generation Tensor Cores also enable substantial speedups in mixed-precision operations, with reports indicating up to 1.3x–1.9x higher throughput for TF32 and FP16 training compared to the preceding RTX 3090 generation [32]. To fully leverage these Tensor Cores, however, matrix dimensions must often be multiples of 8, which can impose constraints on model configurations and batch sizes [1].

When scaling out to an 8-GPU RTX 4090 cluster, the architectural limitation becomes even more pronounced. Without NVLink, inter-GPU communication is relegated to the PCIe bus, which offers significantly lower bandwidth (~16 GB/s per link) and higher latency compared to NVLink (~300 GB/s) [9] [24] [57]. This transforms the cluster from a unified, high-memory pool into a collection of nodes connected by a relatively slow network. Consequently, parallelism strategies that rely heavily on frequent communication between GPUs become severely bottlenecked. Data Parallelism (DP), where each GPU maintains a full copy of the model, quickly becomes unviable as the memory overhead per GPU would exceed 24 GB for any non-trivial model [21]. Instead, sharded parallelism frameworks like Fully-Sharded Data Parallel (FSDP) or DeepSpeed ZeRO become non-negotiable necessities [21] [56]. These frameworks distribute shards of model parameters, gradients, and optimizer states across all available GPUs, effectively aggregating the VRAM of the entire cluster into a single, accessible pool [19]. For example, an 8x RTX 4090 system can behave as a unified ~192 GB VRAM environment, making it possible to train models like Llama-3-8B that would otherwise be impossible on a single GPU [19]. However, this comes at the cost of increased communication volume. Operations like reducing gradients across all devices become a significant source of latency, directly impacting training throughput [6]. This reality makes tensor parallelism, which splits layers across multiple GPUs, impractical for groups larger than two cards, as the communication overhead would dominate the computation [26]. The low interconnect bandwidth also means that the overall efficiency and scalability of the cluster will be substantially lower than on systems with high-bandwidth interconnects, requiring compensatory measures like larger gradient accumulation

steps to achieve stable training [23]. Practitioners must also be aware of potential software-level challenges; early documentation noted that multi-GPU support was not always reliable, and specific issues like NCCL P2P communication failures have been reported, sometimes requiring configuration changes like setting `NCCL_P2P_DISABLE=1` to resolve initialization deadlocks [28] [67]. Therefore, the path to harnessing the power of an 8x RTX 4090 cluster is less about raw hardware aggregation and more about navigating the complex interplay of sharded parallelism, communication bottlenecks, and software configuration nuances.

Single-GPU Optimization on the RTX 4090: Mastering VRAM and Throughput

Operating within the confines of a single NVIDIA RTX 4090, with its 24 GB of VRAM, demands a meticulous and strategic approach to hyperparameter tuning, where every byte of memory and every cycle of compute is accounted for. The primary objective is to maximize the utilization of the available VRAM while maintaining training stability and achieving competitive throughput. This involves a hierarchy of decisions, starting with the choice of model and precision, followed by the careful orchestration of batch size and gradient accumulation, and finally the application of advanced software optimizations to reclaim memory and accelerate computation. The journey begins with acknowledging the fundamental VRAM ceiling, which dictates that full fine-tuning of models exceeding approximately 7 billion parameters is infeasible without significant intervention [23] [71]. For smaller models (e.g., <7B parameters), full fine-tuning in 16-bit precision (BF16 or FP16) is generally achievable, leaving ample room for experimentation with batch size and learning rates [59]. However, for the vast majority of modern transformers and large-scale models, quantization emerges as the indispensable gateway to training on consumer hardware. Quantization-based PEFT methods, most notably QLoRA, drastically reduce the memory footprint by compressing the base model's weights to 4-bit precision and applying LoRA adapters to only a fraction of the trainable parameters [36] [71]. This technique has been shown to reduce the memory required per parameter from 96 bits (full fine-tune in BF16/FP16) to just 5.2 bits, representing an 18-fold reduction that enables the fine-tuning of 7B models on a single RTX 4090 and even facilitates QLoRA of 70B+ models [17] [36].

Once a suitable strategy like QLoRA is chosen, the trainer must configure the core training loop parameters. The per-device batch size (`per_device_train_batch_size`) and the number of gradient accumulation steps (`gradient_accumulation_steps`) are intrinsically linked and represent the primary knobs for controlling the global batch size while respecting the VRAM limit [60]. When VRAM constraints force a small per-device batch size, gradient accumulation allows the model to effectively "see" a much larger dataset before updating its weights. The relationship is inverse: reducing the batch size by a factor of two requires doubling the number of accumulation steps to maintain the same effective batch size [60]. For example, if a trainer finds that a batch size of 4 causes an out-of-memory (OOM) error but a batch size of 2 is manageable, they can increase the gradient accumulation steps from 8 to 16 to compensate [60]. This trade-off, however, comes with consequences. Smaller micro-batches can lead to noisier gradient estimates, potentially affecting convergence stability, while a higher number of accumulation steps increases the total number of iterations per epoch, thereby extending the overall training time [49]. Empirical evidence suggests that for QLoRA fine-tuning of 7B models, a per-device batch size of 2 or 4 with 4 to 8 gradient accumulation steps is a common and effective starting point [60] [62]. For very small models trained in full precision, larger batch sizes like 8, 16, or even 32 may be feasible, allowing for fewer accumulation steps or even zero, which can improve training throughput [45] [46].

Beyond managing batch size, a suite of software-level optimizations is crucial for maximizing VRAM efficiency and computational throughput. Gradient checkpointing is one of the most impactful techniques, as it trades compute for memory by selectively re-computing activations during the backward pass instead of storing them throughout the forward pass [16] [62]. This is particularly beneficial for large transformer models where activation storage can be a major memory bottleneck, albeit at the cost of increased FLOPs [26]. Another transformative optimization is FlashAttention, which replaces the standard PyTorch implementation of scaled dot-product attention with a highly optimized kernel that dramatically reduces both memory usage (scaling linearly vs. quadratically with sequence length) and computation time [17] [48]. Enabling FlashAttention can unlock significantly larger batch sizes and sequence lengths on the same hardware. Finally, using paged optimizers like `paged_adamw_32bit` can offer a modest reduction in VRAM overhead compared to standard AdamW implementations, further aiding memory-constrained scenarios [63] [65]. Collectively, these strategies form a powerful toolkit that allows a skilled practitioner to push the boundaries of what is possible on a single RTX 4090, transforming a memory-bound task into a computationally

bound one where throughput can be maximized through careful tuning of learning rates and parallel processing.

Distributed Training on an 8x RTX 4090 Cluster: Navigating the PCIe Bottleneck

Deploying an 8-GPU cluster of NVIDIA RTX 4090s presents a paradigm shift in training strategy, moving from a single-resource problem to a complex distributed computing challenge dominated by communication constraints. The defining characteristic of this hardware configuration—the complete absence of NVLink or peer-to-peer GPU communication—forces a departure from simple memory aggregation and towards sophisticated sharded parallelism frameworks [9] [25] [31]. Unlike high-end datacenter clusters where GPUs are interconnected with ultra-high-bandwidth links like NVLink (offering ~300 GB/s) or InfiniBand, the 8x RTX 4090 cluster relies solely on the PCIe bus for inter-GPU communication, which provides a much lower bandwidth of approximately 16 GB/s per link [9] [24] [57]. This architectural decision renders traditional approaches like naive Data Parallelism (DP), where each GPU maintains a full replica of the model, completely untenable for all but the smallest models, as the VRAM overhead per GPU would far exceed the 24 GB limit [21]. The aggregate VRAM of 192 GB is thus not directly usable as a contiguous pool but must be managed through explicit sharding.

Consequently, the cornerstone of any viable training configuration on an 8x RTX 4090 cluster is a sharded parallelism framework such as Fully-Sharded Data Parallel (FSDP) or DeepSpeed Zero Redundancy Optimizer (ZeRO) [21] [56]. These frameworks solve the VRAM problem by partitioning the model's parameters, gradients, and optimizer states across all eight GPUs. In FSDP, for example, each parameter is stored on only one device, and gradients are reduced across all devices before being applied locally, ensuring that the memory footprint per GPU scales with the shard size rather than the full model size [55]. This allows the cluster to function as a unified ~192 GB VRAM environment, making it feasible to train large models like Llama-3-8B that would be impossible on a single card [19]. However, this solution introduces a new and critical bottleneck: communication. Every synchronization step—be it averaging gradients or updating optimizer states—requires data to be moved across the PCIe interconnects. This communication overhead can severely limit the overall training throughput, especially for models where the

computational workload per sample is not sufficiently large to hide the latency of these collective operations [23] [26]. As a result, training on an 8x RTX 4090 cluster will inherently be slower than on a similarly sized cluster of A100/H100 GPUs with NVLink, and the benefits of adding more GPUs diminish rapidly beyond a certain point due to diminishing returns from communication costs [8] [24].

Given this communication-centric reality, the configuration of hyperparameters must prioritize minimizing the frequency and volume of these costly operations. This often leads to larger micro-batch sizes and corresponding increases in gradient accumulation steps to maintain a stable global batch size. For instance, a common configuration might involve a `per_device_train_batch_size` of 1 or 2, with `gradient_accumulation_steps` set to 8 or 16, resulting in an effective global batch size of 8 or 32 across the 8 GPUs [49] [56]. While this increases the number of iterations, it helps to amortize the communication cost over more samples. The choice of parallelism stage in frameworks like DeepSpeed ZeRO or FSDP also introduces important trade-offs. ZeRO Stage 3, which shards parameters, gradients, and optimizer states, offers the best VRAM efficiency but incurs the highest communication overhead. Offloading techniques, such as `fsdp_offload_params=true`, can move parts of the optimizer state to CPU memory to free up additional VRAM, but this comes at a severe penalty to training speed [22] [56]. Trainers must therefore carefully balance memory savings against the performance degradation caused by offloading. Furthermore, practitioners must be prepared to address software-level complexities. There have been documented reports of multi-GPU training hangs on RTX 4090 systems due to NCCL P2P communication failures, which can sometimes be resolved by disabling P2P explicitly (`NCCL_P2P_DISABLE=1`) [67]. Compatibility with specific versions of CUDA, PyTorch, and the NVIDIA driver is also critical, as mismatched versions can lead to non-recoverable kernel crashes or unstable behavior, making it essential to use well-tested library stacks [12] [67]. Ultimately, leveraging an 8x RTX 4090 cluster successfully is less about brute-force VRAM pooling and more about astutely managing the delicate balance between computation, memory sharding, and the pervasive bottleneck of PCIe communication.

Numerical Precision and Stability: The Critical Choice Between BF16 and FP16

The selection of numerical precision is one of the most consequential decisions a model trainer makes, with direct impacts on training stability, memory usage, and computational throughput, particularly on a hardware platform like the NVIDIA RTX 4090. The two dominant 16-bit formats, BFloat16 (BF16) and Float16 (FP16), offer a compelling compromise between the memory and speed benefits of reduced precision and the numerical stability of 32-bit floating-point arithmetic. Both formats are natively supported by the RTX 4090's Tensor Cores, enabling significant acceleration in mixed-precision training workflows [1] [41]. However, their underlying bit structures lead to different behaviors, especially in the context of large language models and transformer architectures. BF16 shares the same 8-bit exponent field as 32-bit float (FP32), giving it a wide dynamic range that can represent very small numbers down to approximately 1e-38 without underflowing to zero [2] [42]. This characteristic is a key advantage over FP16, which has a much narrower exponent range (minimum positive normal value of ~6e-8). Consequently, BF16 is less prone to gradient underflow, where small gradients are rounded to zero, corrupting the optimization process [42]. Because of this, mixed-precision training with BF16 typically does not require the loss scaling mechanism that is essential for stable FP16 training, simplifying the training pipeline [1] [42].

Despite these theoretical advantages, empirical evidence has revealed a critical and counterintuitive instability associated with using BF16 in transformer attention mechanisms. Research has demonstrated that the biased rounding errors inherent in BF16 addition can accumulate during the computation of the attention output, leading to a systematic negative bias that triggers a catastrophic loss explosion during training [2] [37]. This issue is particularly prevalent in FlashAttention implementations and manifests when repeated maximum values occur in the pre-softmax scores, causing softmax probabilities to equal exactly 1 and triggering the problematic rounding behavior [2]. This instability has been observed on various hardware platforms, including the NVIDIA A100 and RTX 4090, indicating it is a property of the BF16 format itself in this specific context [37]. The consequence is that while a trainer might switch to BF16 expecting greater stability, they could inadvertently introduce a severe bug that halts training entirely [38]. This creates a significant dilemma: BF16 offers superior dynamic range and eliminates the need for loss scaling, yet it carries a risk of instability that is not present in FP16. This has led to recommendations to either revert to FP16 with proper loss scaling, use a

modified version of FlashAttention designed to mitigate the bias, or fall back to the standard PyTorch attention implementation (`attn_implementation="eager"`), which appears to be more numerically robust under BF16[37][40].

Given this precarious situation, establishing a stable training baseline is paramount. The recommended approach is to start with a conservative and proven configuration. For many transformer models, this means beginning with FP16 mixed-precision training. This requires the use of a `GradScaler` from `torch.cuda.amp` to dynamically or statically scale the loss, preventing gradient underflow[1][11]. The initial loss scaling factor can be set between 8 and 32,768, with dynamic scaling being a popular choice that automatically adjusts the scale based on whether overflow occurred in the previous step[1]. Regardless of the precision used, another critical stability measure is gradient clipping. Applying global norm clipping, which rescales the entire gradient vector if its L2 norm exceeds a predefined threshold, is a standard practice in transformer training to prevent exploding gradients[39][44]. Commonly used thresholds range from 0.3 to 1.0[43][47]. Combining a robust loss scaling mechanism (for FP16) or relying on BF16's wider range (while monitoring for instability) with consistent gradient clipping forms the bedrock of a stable training process. Only after a stable baseline is established should a trainer experiment with BF16, carefully validating the results and implementing mitigations like the safe softmax modification if instability is encountered[37]. This cautious, evidence-based approach is the most prudent way to navigate the complex trade-offs between the theoretical benefits and practical pitfalls of different 16-bit precision formats on the RTX 4090 platform.

Actionable Configuration Tables: A Trainer's Decision-Making Framework

This section consolidates the preceding analysis into two comprehensive tables designed to serve as a practical decision-making framework for configuring ConvNeXt training runs on the specified hardware. The tables are structured to guide the trainer through a logical prioritization of hyperparameters, starting with the most critical decisions regarding model size, strategy, and parallelism, and then providing concrete values for secondary parameters. It is crucial to understand that these configurations are starting points, not final solutions. The optimal settings are highly dependent on the specific model architecture, dataset characteristics, and

performance goals. Therefore, empirical validation remains the ultimate arbiter of success, and trainers are encouraged to treat these tables as a foundation upon which to build and refine their own experiments.

The first table provides guidance for training on asingle NVIDIA RTX 4090 GPU. The primary axis of organization is model size, as this dictates the fundamental strategy required to manage the 24 GB VRAM constraint. For smaller models (<7B parameters), full fine-tuning is often feasible, allowing for more flexibility in batch size. For medium-sized models (7B-13B), QLoRA is the standard approach, necessitating a focus on balancing the `per_device_train_batch_size` with `gradient_accumulation_steps`. For very large models (>30B), QLoRA is virtually mandatory, and the trainer must ensure that all relevant flags for quantization and sharding are correctly configured. Each row provides a suggested set of parameters, along with notes explaining the rationale behind the choices, referencing the architectural constraints and optimization techniques discussed previously.

Model Size (Billions)	Strategy	Precision	per_device_train_batch_size	gradient_accumulation_steps	enable_gradient_checkpointing	
< 7	Full Fine-Tuning	bf16	8	1	True	
7 - 13	QLoRA	bf16	4	4	True	
13 - 70	QLoRA + FSDP	bf16	2	8	True	

Model Size (Billions)	Strategy	Precision	per_device_train_batch_size	gradient_accumulation_steps	enable_gradient_checkpointing	
> 70	QLoRA + FSDP + Offload	bf16	1	16	True	

The second table addresses training on an8x NVIDIA RTX 4090 cluster. Here, the organizing principle shifts from model size to the parallelism strategy, as sharded frameworks like FSDP and DeepSpeed ZeRO are non-negotiable for all but the smallest models. The key insight is that the cluster's VRAM is aggregated, but its performance is limited by the slow PCIe interconnect. This reality influences the choice of sharding level and the necessity of using larger micro-batches and gradient accumulation steps to amortize communication costs. The table provides examples for both FSDP and DeepSpeed, highlighting their respective features and trade-offs, such as the ability of DeepSpeed to offload parameters to CPU memory. Trainers must also be mindful of potential software-level quirks, such as the need to set `NCCL_P2P_DISABLE=1` in some cases to avoid initialization deadlocks on multi-RTX 4090 systems[67].

Strategy	Model Size (Billions)	Precision	per_device_train_batch_size	gradient_accumulation_steps	Sharding / Offloading Level	offlo
FSDP	**Any**	**bf16**	**2**	**8**	FULL_SHARD	**False**
DeepSpeed ZeRO Stage 3	**Any**	**bf16**	**1**	**16**	ZERO_STAGE=3	**False**
DeepSpeed ZeRO Stage 3 w/ CPU Offload	**Any**	**bf16**	**1**	**16**	ZERO_STAGE=3, OFFLOAD_OPTIMIZER=True	**True**

In summary, these tables provide a structured pathway for navigating the unique challenges and opportunities presented by the NVIDIA RTX 4090 hardware. By understanding the fundamental constraints—VRAM limits on a single card and PCIe bottlenecks in a cluster—and systematically applying the appropriate strategies, trainers can optimize their ConvNeXt models for both memory efficiency and computational performance.

Reference

1. **Train With Mixed Precision** https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html
2. **Why Low-Precision Transformer Training Fails: An Analysis ...** https://arxiv.org/html/2510.04212v2
3. **The Complete Guide to Fine-Tuning LLMs and SLMs in 2025** https://medium.com/@nraman.n6/the-complete-guide-to-fine-tuning-llms-and-slms-in-2025-75087978fc6e
4. **(LoRA) Fine-Tuning FLUX.1-dev on Consumer Hardware** https://huggingface.co/blog/flux-qlora
5. **NanoGPT Speedrun Living Worklog - Tyler Romero** https://www.tylerromero.com/posts/nanogpt-speedrun-worklog/
6. **Better Exploiting First Attentions for Efficient Transformer ...** https://arxiv.org/html/2510.14614v2
7. **ModernBERT, the BERT of 2024 - Gonzo ML** https://gonzoml.substack.com/p/modernbert-the-bert-of-2024
8. **Choose The Right Number Of GPUs For Deep Learning ...** https://acecloud.ai/blog/gpu-requirement-for-deep-learning-workstation/
9. **Dual RTX 4090 with distributed training** https://forums.developer.nvidia.com/t/dual-rtx-4090-with-distributed-training/289454
10. **CUDA OutOfMemoryError with 10 Images on RTX 4090 ...** https://github.com/facebookresearch/vggt/issues/31
11. **GPU memory consumption increases while training** https://discuss.pytorch.org/t/gpu-memory-consumption-increases-while-training/2770
12. **Faster R-CNN training failure on RTX 4090 (CUDA 12.4)** https://www.technetexperts.com/faster-rcnn-rtx-4090-training-crash-fix/

13. **Force GPU memory limit in PyTorch** https://stackoverflow.com/questions/49529372/force-gpu-memory-limit-in-pytorch
14. **System Memory Optimization for SSD-Offloaded LLM Fine- ...** https://arxiv.org/html/2505.23254
15. **Looking for ways to calculate max batch size supported by ...** https://forums.developer.nvidia.com/t/looking-for-ways-to-calculate-max-batch-size-supported-by-any-given-gpu-for-model-training/306658
16. **GPU Survival Guide: Avoid OOM Crashes for Large Models** https://www.runpod.io/articles/guides/avoid-oom-crashes-for-large-models
17. **hiyouga/LLaMA-Factory: Unified Efficient Fine-Tuning of ...** https://github.com/hiyouga/LLaMA-Factory
18. **Fine-tuning the LLaMA 2 model on RTX 4090** https://vast.ai/article/Fine-tuning-the-LLaMA-2-model-on-RTX-4090
19. **FSDP, DeepSpeed and Accelerate - Parlance Labs** https://parlance-labs.com/education/fine_tuning/zach.html
20. **Messing around with fine-tuning LLMs, part 4 -- training ...** https://www.gilesthomas.com/2024/05/fine-tuning-4
21. **Everything about Distributed Training and Efficient Finetuning** https://sumanthrh.com/post/distributed-and-efficient-finetuning/
22. **Efficiently fine-tune Llama 3 with PyTorch FSDP and Q-Lora** https://www.philschmid.de/fsdp-qlora-llama3
23. **GPU Options for Finetuning Large Models** https://www.digitalocean.com/resources/articles/gpu-options-finetuning
24. **GPU Benchmarks of H100/H200/A100/RTX 4090 ... - WhaleFlux** https://www.whaleflux.com/blog/gpu-benchmarks-of-h100-h200-a100-rtx-4090-and-whaleflux-resource-management-solution/
25. **Everything You Need to Know About the Nvidia RTX 4090 ...** https://www.runpod.io/articles/guides/nvidia-rtx-4090
26. **Why A100/H100 Beat RTX 4090 for Large-Model Training** https://www.allpcb.com/allelectrohub/why-a100h100-beat-rtx-4090-for-large-model-training
27. **RTX 4090 vs H100: Performance and Cost Breakdown | Blog** https://computeprices.com/blog/rtx-4090-vs-h100-performance-and-cost-breakdown
28. **Deep Learning GPU Benchmarks 2022 - aime.info** https://www.aime.info/blog/en/deep-learning-gpu-benchmarks-2022/
29. **The Best GPUs for Deep Learning in 2023** https://timdettmers.com/2023/01/30/which-gpu-for-deep-learning/

30. **MSPO: A machine learning hyperparameter optimization ...** https://pmc.ncbi.nlm.nih.gov/articles/PMC12277558/
31. **RTX 4090 VS dual RTX 3090 for deep learning build?** https://www.reddit.com/r/deeplearning/comments/185gpiv/rtx_4090_vs_dual_rtx_3090_for_deep_learning_build/
32. **NVIDIA RTX 4090: A Powerhouse for Deep Learning and AI** https://www.poolcompute.com/blog/nvidia-rtx-4090-for-deep-learning-and-ai
33. **RTX 4090 VRAM: Is 24GB Enough for Gaming and Content ...** https://groovycomputers.ca/blogs/resources/4090-vram?srsltid=AfmBOopbNcjYDIltcBmtLcdSYiFKC4Divddbljogvrpf XeoJbllsnHeM
34. **Best GPU for machine learning | Blog** https://northflank.com/blog/best-gpu-for-machine-learning
35. **NVIDIA GeForce RTX 4090 : Architecture, Working, & Its Uses** https://www.elprocus.com/nvidia-geforce-rtx-4090/
36. **The Ultimate Guide to Fine-Tuning LLMs from Basics ...** https://arxiv.org/html/2408.13296v1
37. **Why Low-Precision Transformer Training Fails: An Analysis ...** https://arxiv.org/html/2510.04212v1
38. **When llama uses bf16 training, there is an abnormal loss** https://github.com/EleutherAI/gpt-neox/issues/947
39. **Gradient Clipping: Keeping Your Training From Blowing Up** https://satyamcser.medium.com/gradient-clipping-keeping-your-training-from-blowing-up-ff353e5294e0
40. **Gradient Overflow issue while using deepspeed - Beginners** https://discuss.huggingface.co/t/gradient-overflow-issue-while-using-deepspeed/167833
41. **Stabilizing LLM Training: Techniques and Insights** https://www.rohan-paul.com/p/stabilizing-llm-training-techniques
42. **Why bf16 do not need loss scaling? - mixed-precision** https://discuss.pytorch.org/t/why-bf16-do-not-need-loss-scaling/176596
43. **Fixing the Exploding Gradient Problem - Deep Learning** https://forums.fast.ai/t/fixing-the-exploding-gradient-problem/86277
44. **Understanding Gradient Clipping (and How It Can Fix ...** https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradients-problem
45. **Hybrid sequence learning with interpretability for multi- ...** https://www.sciencedirect.com/science/article/pii/S2590123025024788

46. **Spatio-temporal transformer traffic prediction network ...** https://pmc.ncbi.nlm.nih.gov/articles/PMC12404550/
47. **Transformer-Based Re-Ranking Model for Enhancing ...** https://www.mdpi.com/2079-9292/14/2/243
48. **Dao-AILab/flash-attention: Fast and memory-efficient ...** https://github.com/Dao-AILab/flash-attention
49. **Building Domain-Specific Small Language Models via ...** https://arxiv.org/html/2511.21748v1
50. **Sensitivity Analysis for Deep Learning: Ranking Hyper- ...** https://www.researchgate.net/publication/357236505_Sensitivity_Analysis_for_Deep_Learning_Ranking_Hyper-parameter_Influence
51. **Accelerating LLMs with llama.cpp on NVIDIA RTX Systems** https://developer.nvidia.com/blog/accelerating-llms-with-llama-cpp-on-nvidia-rtx-systems/
52. **Llama 3.1 8B GPU benchmark – $0.228 per Million output ...** https://blog.salad.com/llama-3-1-8b/
53. **LLM Inference - Consumer GPU performance** https://www.pugetsystems.com/labs/articles/llm-inference-consumer-gpu-performance/?srsltid=AfmBOor5FAPnLvFsYd3LY9zijR4sR6LYo9X_JAlsfPrjdSdrkvaTwr_H
54. **Maximizing training throughput using PyTorch FSDP** https://pytorch.org/blog/maximizing-training/
55. **Benchmarking Advanced Multi‑GPU Training Strategies** https://medium.com/@savyasachi.thati/benchmarking-advanced-multi-gpu-training-strategies-20c9675003db
56. **Distributed Training - LLaMA Factory** https://llamafactory.readthedocs.io/en/latest/advanced/distributed.html
57. **Deep Learning Server | Form and Formula** https://blbadger.github.io/gpu-server.html
58. **Fira: Can We Achieve Full-rank Training of LLMs under ...** https://arxiv.org/html/2410.01623v3
59. **Efficient Fine-Tuning for Llama-v2-7b on a Single GPU** https://www.linkedin.com/posts/nkvn_efficient-fine-tuning-for-llama-v2-7b-on-activity-7102401563687600129-upIQ
60. **Fine-Tuning LLaMA 2: A Practical Guide | by whyamit404** https://medium.com/@whyamit404/fine-tuning-llama-2-a-practical-guide-8c1d262693cd
61. **Fine-Tuning Llama-2: A Comprehensive Case Study ...** https://news.ycombinator.com/item?id=37090632

62. **A poor man's guide to fine-tuning Llama 2** https://duarteocarmo.com/blog/fine-tune-llama-2-telegram.html
63. **How to Fine-Tune LLaMA-2 on Your Own Data at Scale** https://predibase.com/blog/how-to-fine-tune-llama-2-on-your-data-with-scalable-llm-infrastructure
64. **GPU run out of memory while fine tuning Llama 2 with 2 ...** https://stackoverflow.com/questions/78068912/gpu-run-out-of-memory-while-fine-tuning-llama-2-with-2-rtx-4090
65. **Fine-Tuning LLaMA 2: A Step-by-Step Guide to ...** https://www.datacamp.com/tutorial/fine-tuning-llama-2
66. **A Comprehensive Guide to Fine-Tuning in Google Colab** https://medium.com/artificial-corner/mastering-llama-2-a-comprehensive-guide-to-fine-tuning-in-google-colab-bedfcc692b7f
67. **RTX 4090 Huggingface Trainer Compatible? - Intermediate** https://discuss.huggingface.co/t/rtx-4090-huggingface-trainer-compatible/25032
68. **meta-pytorch/torchtune: PyTorch native post-training library** https://github.com/meta-pytorch/torchtune
69. **Finetune LLMs on your own consumer hardware using ...** https://pytorch.org/blog/finetune-llms/
70. **Fine-Tuning Your First Large Language Model (LLM) with ...** https://huggingface.co/blog/dvgodoy/fine-tuning-llm-hugging-face
71. **The Ultimate Guide to Fine-Tuning LLMs from Basics ...** https://arxiv.org/html/2408.13296v2?ref=chitika.com