Bandit-Guided NCCL Tuning for Multi-Node GPU Clusters

Varun Madan
Stanford University
Department of Computer Science
Stanford, CA, USA
vmadan@stanford.edu

Ashley Raigosa
Stanford University
Department of Computer Science
Stanford, CA, USA
raigosa@stanford.edu

Abstract

NVIDIA's NCCL library is the de facto standard for collective communication in distributed deep learning, relying on a built-in cost model (AUTO) to select the algorithm and protocol for each collective at runtime. We present a systematic characterization of AUTO's selection quality across three cluster topologies (1×8, 2×4, and 4×2 A100 GPUs) under both isolated and compute-overlapped workloads. We find two compounding sources of suboptimality. First, compute overlap changes the optimal algorithm–protocol pair at 6 of 9 message sizes on a single NVLink-connected node, because protocols differ in streaming multiprocessor (SM) consumption and compete with concurrent compute kernels. This effect is invisible to NCCL's cost model. Second, these gaps amplify across nodes: on a 2-node cluster, AUTO selects Ring for every message size even though Tree delivers up to 2.9× higher throughput, producing a 57% latency penalty at 64 MB. We initially address this with a profile-guided offline tuner, but find it brittle in cloud environments where preemption invalidates the profiled policy. To provide robust adaptation, we present an online reinforcement-learning bandit tuner that learns the optimal configuration through deterministic round-robin exploration using NCCL's tuner plugin API. Across multi-node topologies, the bandit achieves 25–42% latency reductions over AUTO with a safety gate that prevents regressions, requiring only 40 exploration iterations before converging.

CCS Concepts

• **Networks** → **Network performance evaluation**; • **Computing methodologies** → *Parallel algorithms*; *Online learning settings*.

Keywords

NCCL, collective communication, GPU clusters, multi-armed bandit, performance tuning

ACM Reference Format:
Varun Madan and Ashley Raigosa. 2026. Bandit-Guided NCCL Tuning for Multi-Node GPU Clusters. In *Proceedings of Stanford CS244C: Advanced Networking and Distributed Systems (CS244C '26).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

CS244C '26, Stanford, CA

ACM ISBN 978-1-4503-XXXX-X/2026/03
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Distributed deep learning at scale depends on efficient collective communication. As models have grown from millions to hundreds of billions of parameters [2, 4, 12, 14], gradient synchronization via AllReduce has become a critical bottleneck, often consuming a significant fraction of total training time [7]. NVIDIA's Collective Communications Library (NCCL) [10] is the de facto standard, providing optimized AllReduce, AllGather, and ReduceScatter primitives that support multiple *algorithms* (Ring, Tree) and *protocols* (Simple, LL128, LL). NCCL's AUTO mode selects the algorithm–protocol pair for each collective using an internal cost model, and practitioners rarely override this default.

However, AUTO's cost model is *static*: it derives bandwidth estimates from the detected topology at communicator initialization, calibrated for isolated collective operations on NVLink. Two assumptions underlying this model break down in production training. First, modern training pipelines overlap gradient AllReduce with backward-pass compute on concurrent CUDA streams [6, 7], introducing SM contention that changes the relative performance of protocols. Second, multi-node clusters introduce heterogeneous link hierarchies (NVLink intra-node, TCP/RDMA inter-node) that dramatically alter algorithm efficiency, yet the cost model is effectively calibrated for isolated execution and does not capture heterogeneous multi-node link effects.

Through systematic experiments across three topologies (1×8, 2×4, and 4×2 A100 GPUs, all world size 8), we discover two compounding sources of suboptimality in AUTO's selections:

- **Overlap sensitivity**: On a single NVLink-connected node, 6 of 9 message sizes change their optimal configuration when concurrent compute is present, because protocols differ in SM footprint and contend differently with the compute kernel.
- **Multi-node amplification**: AUTO's modest single-node gap (mean 1.2%) amplifies to 57% at 64 MB on a 2-node cluster, where it selects Ring even though Tree is 2.9× faster. Ring forces every data element to traverse the slow inter-node link at every hop; Tree exploits the intra/inter-node hierarchy.

We first build a *profile-guided offline tuner* that benchmarks each cluster at startup and applies a measured-optimal policy through NCCL's tuner plugin API [9]. A 5-gate statistical framework ensures only high-confidence overrides. However, we discover this approach is *brittle* in cloud environments: when the provider preempts and reassigns containers to different physical nodes, the profiled policy becomes stale and causes regressions.

This motivates our main contribution: an *online RL bandit tuner* that learns the optimal configuration during training through deterministic round-robin exploration. By treating algorithm–protocol

selection as a multi-armed bandit problem [1], the tuner adapts to the actual deployed cluster without offline profiling. A 5% safety gate prevents regressions by falling back to AUTO when the improvement is within noise. Across multi-node topologies, the bandit achieves 25–42% latency reductions over AUTO, converging in 40 iterations (~10 seconds of training time).

Our contributions are:

(1) A characterization showing compute overlap changes optimal NCCL selections at 6/9 message sizes, and multi-node amplifies AUTO's gap from 1.2% to 57%.
(2) An analysis of *why* AUTO fails: its cost model is calibrated for isolated collectives and does not account for SM contention under overlap or heterogeneous link hierarchies across nodes.
(3) An online RL bandit tuner using NCCL's plugin API with deterministic exploration and a safety gate, implemented in 433 lines of C.
(4) An evaluation showing 25–42% improvements on multi-node topologies with zero regressions.

2 Background

2.1 NCCL Algorithms and Protocols

AllReduce computes an element-wise reduction (e.g., sum of gradients) across all ranks and distributes the result to every rank, combining a reduce and a broadcast in one collective. NCCL implements two primary algorithms for AllReduce. **Ring** arranges ranks in a logical ring and pipelines data through $N-1$ reduce-scatter and $N-1$ allgather steps (for N ranks), requiring every data chunk to visit every rank. **Tree** uses a hierarchical reduce-broadcast pattern where data travels $O(\log N)$ hops: ranks reduce toward a root, then broadcast the result. On a homogeneous NVLink mesh, Ring and Tree achieve similar bandwidth, but on heterogeneous topologies, Tree can exploit the link hierarchy by confining high-volume traffic to fast intra-node links.

Each algorithm can use one of three protocols. **Simple** performs large-chunk GPU-initiated DMA transfers: it briefly occupies SMs to set up each transfer, then relies on NVLink hardware to move data, freeing SMs between chunks. **LL128** uses 128-byte flag-based transfers that tend to impose higher sustained SM pressure through polling and flagging logic. **LL** uses 8-byte atomic units with similar continuous SM occupancy but at lower throughput. The protocol choice determines not just peak SM allocation but also the *sustained* SM utilization pattern, which directly affects how much compute capacity remains for concurrent kernels.

2.2 The Tuner Plugin API

Since NCCL v2.19, an external *tuner plugin* [9] can override AUTO's selections. The plugin exposes two callbacks. `pluginInit(nRanks, nNodes, ...)` is called once per communicator creation to provide the topology, and `pluginGetCollInfo(collType, nBytes, collCostTable, ...)` is called before each collective to pass the message size and a cost table that the plugin may modify. By setting a cost of zero for the desired (algorithm, protocol) pair, the plugin forces NCCL to select it. Crucially, the plugin has no visibility into concurrent compute workloads and no history of prior iterations; it sees only the current collective's parameters.

2.3 Compute–Communication Overlap

Modern training frameworks overlap gradient AllReduce with backward-pass computation [7, 12]. When both streams compete for SMs, protocols that sustain higher SM pressure [5] may slow concurrent compute, increasing total iteration time *even if the collective itself is faster in isolation.* NCCL's cost model cannot account for this because it evaluates each collective independently, without knowledge of other active CUDA streams.

3 Characterization

We profile AllReduce (fp32) across 5 configurations (AUTO, Tree+Simple, Tree+LL128, Ring+Simple, Ring+LL128) at 9 message sizes (32 KB–256 MB) on 8× A100-SXM4-80GB GPUs provisioned through Modal cloud. Each measurement consists of 50 iterations after 10 warmup iterations; we report medians. Overlap mode runs a concurrent 4096×4096 fp32 matmul on a separate CUDA stream, simulating backward-pass compute. We test three topologies: 1×8 (single-node, full NVLink mesh), 2×4 (2 nodes, 4 GPUs each), and 4×2 (4 nodes, 2 GPUs each).

Our message sizes span 32 KB to 256 MB, covering the range of gradient bucket sizes used in distributed training; smaller messages are dominated by launch and synchronization overhead, while larger messages approach steady-state bandwidth.

3.1 Overlap Changes the Optimal Configuration

Table 1 shows single-node results under sequential (isolated) and overlapped execution. Under sequential AllReduce, the optimal configuration varies across sizes: Tree+LL128 wins at small sizes, Ring variants at medium sizes, and AUTO ties at 256 MB. Under overlap, **6 of 9 message sizes change their optimal configuration** (marked "Flip" in the table). The most striking pattern: Ring+Simple dominates for sizes ≥2 MB under overlap, displacing Tree+LL128 and other protocols that won in isolation.

Table 1: Single-node AllReduce results (8× A100 NVLink). Gap% is AUTO overhead vs. best. "Flip" indicates the optimal config changes between sequential and overlap.

Size	Sequential			Overlap			
	Best	ms	Gap	Best	ms	Gap	Flip
32KB	Tree+L	8.33	1.5%	Tree+S	8.32	0.1%	✓
64KB	Ring+L	8.44	0.0%	AUTO	8.12	0.0%	✓
256KB	Ring+S	8.35	3.0%	AUTO	8.36	0.0%	✓
1MB	Tree+L	8.56	0.3%	Tree+L	8.28	1.3%	
2MB	Ring+L	8.45	1.7%	Ring+S	8.60	3.3%	✓
4MB	Tree+S	8.50	0.1%	Ring+S	8.46	2.1%	✓
16MB	Ring+S	8.50	3.1%	Ring+S	8.56	3.0%	
64MB	Ring+S	8.95	2.3%	Ring+S	9.23	0.6%	
256MB	AUTO	10.6	0.0%	Ring+S	10.6	0.4%	✓
Mean gap						1.2%	6/9

The mechanism is *SM contention*: under overlap, NCCL's communication kernels and the compute matmul compete for SM slots. As described in Section 2, Simple's DMA-based transfers briefly occupy SMs for setup but release them between chunks, while LL128 tends

to sustain higher SM pressure through its flag-based polling. Under overlap, this sustained occupancy matters: Ring+Simple's bursty SM usage pattern leaves more compute capacity for the concurrent matmul kernel between transfer phases, reducing total iteration time. Tree+LL128, despite achieving higher raw communication throughput in isolation, sustains higher SM pressure that slows the concurrent compute. Additionally, Tree's hierarchical reduction can increase communication-side SM pressure relative to Ring's simpler pipeline in our setting. The net effect is that *total iteration time* (compute + communication) favors Ring+Simple under overlap, even when Tree+LL128 is faster in an isolated communication benchmark.

Figure 1 visualizes this shift: the optimal configuration at each message size changes between sequential and overlap modes, with the right panel marking the six flips. While AUTO's mean gap on single-node is modest (1.2%), the *qualitative* finding that the winner changes is what motivates workload-aware tuning.

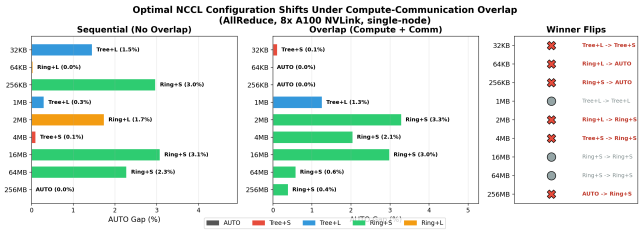

Figure 1: Optimal (algorithm, protocol) for each message size under sequential vs. overlapped execution on single-node 8× A100. The right panel marks winner flips: six of nine sizes change their optimal configuration under overlap.

3.2 Multi-Node Amplifies AUTO's Gap

On single-node NVLink, AUTO's suboptimality is masked by the uniformly high link bandwidth (~25 GB/s per NVLink). When we move to multi-node topologies with heterogeneous links, the gap grows dramatically.

Table 2 shows results on the 2×4 topology (2 nodes, 4 GPUs each, inter-node TCP). Tree+Simple is optimal at every size for both modes, with AUTO gaps reaching **57.2%** at 64 MB sequential and **45.6%** under overlap. Ring algorithms are catastrophically slow: at 64 MB, Ring+Simple takes 337 ms versus Tree+Simple's 117 ms, a **2.9**× slowdown.

Table 2: Multi-node AllReduce results (2×4 A100, inter-node network). Gap% is AUTO overhead vs. best configuration.

Size	SEQ Best (ms)	SEQ Gap	OVL Best (ms)	OVL Gap
256KB	9.5 (T+S)	9.1%	8.8 (T+S)	10.9%
1MB	11.3 (T+S)	13.2%	10.4 (T+S)	13.5%
4MB	15.5 (AUTO)	0.0%	14.1 (T+S)	6.4%
16MB	34.2 (T+S)	15.9%	32.5 (T+S)	11.5%
64MB	116.5 (T+S)	**57.2%**	148.6 (T+S)	**45.6%**
256MB	675.5 (T+S)	0.5%	678.1 (T+L)	0.0%

Figure 2 quantifies the amplification: at 64 MB, AUTO's gap grows from 2.3% on single-node to 57.2% on 2×4, a **25× amplification**. At 1 MB the amplification is 44× (0.3% → 13.2%). The pattern holds across all tested sizes: the gap grows monotonically as the inter-node ratio increases from 1×8 to 2×4 to 4×2.

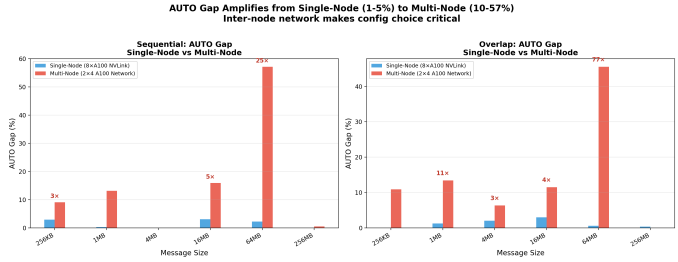

Figure 2: AUTO gap (%) on single-node (1×8) vs. multi-node (2×4). The gap amplifies 3–25× across nodes. At 64 MB, a 2.3% single-node gap becomes 57.2%.

3.3 Why AUTO Fails at Multi-Node

The root cause is topological: NCCL's cost model estimates bandwidth assuming homogeneous links. On NVLink, Ring is efficient because every hop is fast. On multi-node, Ring forces every data chunk through $N-1$ hops, of which a fraction cross the slow inter-node fabric. For 4×2 (4 nodes, 2 GPUs each), Ring requires 7 hops per element, with 3 crossing the inter-node link. Tree, by contrast, reduces within each node using NVLink and sends only one aggregated message per node across the inter-node link, requiring $O(\log_2 N_{\text{nodes}})$ inter-node messages versus $O(N)$ for Ring.

Figure 3 makes this visible: Ring variants (green, yellow) are consistently slower than Tree (red, blue) at every message size, with the gap widening dramatically at large sizes. The rightmost panel summarizes AUTO's suboptimality across all sizes, peaking at 57% (sequential) and 46% (overlap) at 64 MB. The cost model never triggers a switch from Ring to Tree because its bandwidth estimate does not distinguish between intra-node NVLink (600 GB/s aggregate) and inter-node TCP (~10 Gbps). As the inter-node ratio increases, a larger fraction of Ring's hops cross the slow fabric, making its linear topology increasingly pathological relative to Tree's hierarchical approach.

4 Design

Our characterization reveals that AUTO's selections are suboptimal under overlap and lead to severe performance degredation at multi-node scale. We describe two approaches: an offline profiler that measures the optimal policy per cluster (§4.1), and an online bandit that learns during training (§4.2). The offline approach works when the cluster is stable but fails under preemption; the bandit addresses this fundamental limitation.

4.1 Offline Profile-Guided Tuner

The offline tuner profiles the cluster at startup by benchmarking each (algorithm, protocol) pair at 6 key message sizes (256 KB–256 MB), taking approximately 2 minutes. For each size, it applies

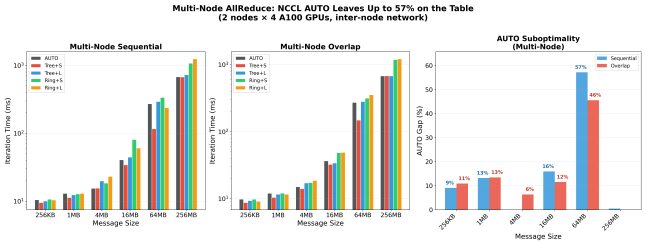

Figure 3: Multi-node AllReduce on 2×4 A100. Left/center: per-config iteration time (log scale) under sequential and overlap modes. Ring variants (green, yellow) are catastrophically slower than Tree (red, blue) at large sizes. Right: AUTO suboptimality reaches 57% at 64 MB.

a 5-gate statistical framework to decide whether to override AUTO: (1) IQR outlier trimming removes measurements outside $[Q_1 - 1.5 \cdot \text{IQR},\ Q_3 + 1.5 \cdot \text{IQR}]$; (2) a one-sided Mann-Whitney U test requires $p < 0.01$; (3) Cliff's delta must exceed 0.33 (medium effect); (4) split-half consistency requires odd and even samples to agree on the winner; (5) the improvement must exceed a size-adaptive threshold (5% for <64 MB, 10% for ≥64 MB). Only when all five gates pass does the tuner override AUTO. The resulting policy is encoded as a comma-separated string and loaded via the tuner plugin's cost-table manipulation.

On the profiled cluster, the offline tuner correctly identifies Tree+Simple for large multi-node messages, matching our characterization. However, in cloud environments (Modal, AWS), training jobs are routinely preempted and rescheduled to different physical nodes. The profiled policy (measured on cluster A) becomes stale when the job resumes on cluster B with different inter-node characteristics. We observed this in practice: a policy profiled on one 2×4 assignment produced a 110% regression after Modal preempted and reassigned the containers to a cluster with faster AUTO performance.

Takeaway: offline profiling is sound when the cluster is stable, but fundamentally brittle in elastic cloud environments. The tuner must adapt *online*.

4.2 Online RL Bandit Tuner

We formulate algorithm–protocol selection as a *multi-armed bandit* [1] problem. The action space is small and discrete (four candidate configurations), and each collective invocation yields an immediate latency measurement, making this a natural fit for the classical bandit setting. We define four arms:

(1) Tree + Simple
(2) Tree + LL128
(3) Ring + Simple
(4) AUTO (no override)

The bandit operates per (collective type, size band, topology) key, where size bands are quantized message-size buckets (e.g., 1–16 MB, 16–128 MB), allowing different selections for different message-size ranges. The reward signal is the observed iteration latency (lower is better), logged by the training script after each AllReduce.

Deterministic exploration. Standard bandit strategies (UCB1, Thompson sampling, ϵ-greedy) use random number generators. In NCCL, the tuner plugin runs independently on every rank. If RNG seeds diverge across ranks, different ranks select different algorithms, a correctness violation since all ranks must agree on the collective algorithm. In our v1 prototype using ϵ-greedy, RNG divergence across ranks caused a −85.8% regression on 2×4 at 64 MB. Switching to deterministic exploration in v2 eliminated the divergence entirely, turning a −85.8% regression into a +42.0% improvement, a swing of nearly 128 percentage points.

We replace RNG with *deterministic round-robin*: iteration i selects arm (i mod 4). All ranks compute the same arm from the global iteration count, requiring no inter-rank communication for agreement. We run M=10 rounds of exploration (10 samples per arm, 40 total iterations), then transition to exploitation.

Exploitation with IQR trimming. At the explore-to-exploit transition, the bandit ingests all logged rewards, computes IQR-trimmed means per arm, and selects the arm with the lowest trimmed mean. The IQR trimming (removing values outside $[Q_1 - 1.5 \cdot \text{IQR},\ Q_3 + 1.5 \cdot \text{IQR}]$) ensures that occasional latency spikes do not bias arm selection. All subsequent iterations exploit the selected arm.

Safety gate. After selecting the exploit arm, the bandit compares its trimmed mean against AUTO's. If the improvement is less than 5%, the bandit keeps AUTO. The risk of a marginal or noisy improvement is not worth the potential regression. This gate is critical: it caught the 256 MB case on 2×4, where the bandit and AUTO were within noise (749.2 ms vs. 749.1 ms).

Implementation. The bandit is implemented as a 433-line C shared library loaded via NCCL_TUNER_PLUGIN. Rank 0 writes a reward log; all ranks read it at the explore-to-exploit transition via a shared filesystem. The implementation has zero dependencies beyond libc and libm.

5 Evaluation

We evaluate the bandit tuner on two multi-node topologies (2×4 and 4×2) at 64 MB and 256 MB, the message sizes where AUTO has the largest gap (Table 2) and which correspond to typical gradient buffer sizes in large-model training. Each scenario runs 240 total iterations (40 explore + 200 exploit) with a concurrent matmul workload. The AUTO baseline runs 240 iterations with no tuner plugin. We report median latency of the exploit phase (for the bandit) and all 240 iterations (for AUTO).

5.1 Headline Results

Table 3 and Figure 4 summarize the results. The bandit achieves substantial improvements in 3 of 4 scenarios:

Table 3: Bandit tuner results. Improvement is median latency reduction vs. AUTO. The safety gate correctly keeps AUTO for 2×4 at 256 MB.

Topo	Size	AUTO (ms)	Bandit (ms)	Improv.	Decision
4×2	64MB	81.8	60.9	**+25.5%**	Bandit
4×2	256MB	251.7	194.6	**+22.7%**	Bandit
2×4	64MB	287.3	166.6	**+42.0%**	Bandit
2×4	256MB	749.1	749.2	−0.0%	AUTO (gate)

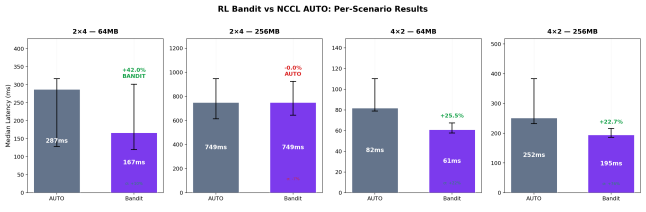

Figure 4: Median AllReduce latency: AUTO vs. bandit tuner. The bandit achieves 25–42% reductions on 3 of 4 scenarios; the safety gate correctly keeps AUTO on 2×4 at 256 MB.

On 4×2, the bandit reduces 64 MB latency from 81.8 ms to 60.9 ms (+25.5%) and 256 MB from 251.7 ms to 194.6 ms (+22.7%). On 2×4, the 64 MB improvement is even larger: 287.3 ms to 166.6 ms (+42.0%). For 2×4 at 256 MB, the safety gate correctly identifies that the bandit's selected arm (749.2 ms) offers no improvement over AUTO (749.1 ms) and falls back, resulting in *zero regression*.

5.2 Variance Reduction and Learning Dynamics

Beyond reducing median latency, the bandit produces *more predictable* iteration times. Figure 5 compares the latency distributions of AUTO and the bandit (exploit-only iterations) across all four scenarios. By committing to a single arm, the bandit avoids the variance introduced by AUTO's cost model switching between suboptimal configurations across iterations.

The effect is pronounced: on 4×2 at 256 MB, the interquartile range narrows by 80.3% and the standard deviation drops from 298 ms to 71 ms, a 76% reduction. On 2×4 at 64 MB, the IQR narrows by 13.7% and standard deviation drops by 19.4%. Even in the 2×4 256 MB case where median improvement is negligible (−0.0%), the bandit still tightens the IQR by 15.0%. More predictable iteration times directly benefit large-scale training [12]: high-variance collective latencies create unpredictable synchronization delays that propagate across data-parallel ranks, wasting GPU cycles.

The bandit's arm selections match our characterization: it selects Tree+Simple at 64 MB on both topologies, consistent with Section 3.2's finding that Tree dominates Ring on multi-node. The 40-iteration exploration phase converges quickly: each arm is tested 10 times with IQR-trimmed means, and the best arm typically emerges within 2–3 rounds.

Exploration cost. The 40 exploration iterations incur a one-time cost of approximately 4–12 seconds (depending on message size). At a per-iteration savings of ~20 ms, the bandit breaks even after approximately 140 iterations. For training runs of thousands of iterations, the exploration cost is negligible.

6 Related Work

Collective communication optimization. TACCL [11] synthesizes topology-aware collective algorithms from communication sketches. TE-CCL [8] formulates collective design as a multi-commodity flow problem. MSCCLang [3] provides a DSL for expressing custom collective schedules. These systems generate *new* algorithms; our bandit selects among NCCL's *existing* algorithms at runtime, making it complementary. A bandit could equally select among

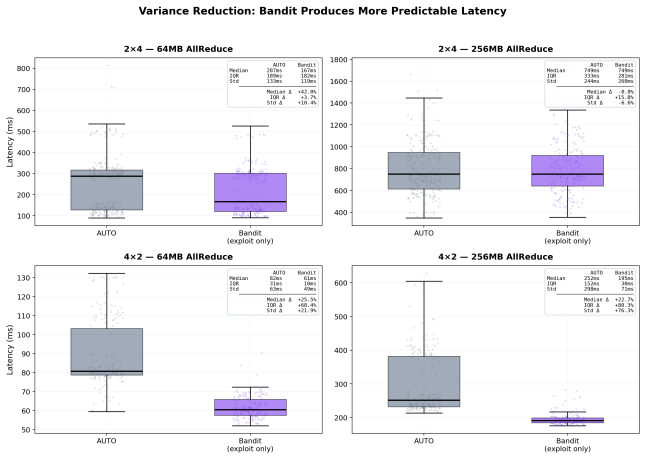

Figure 5: Latency distributions for AUTO vs. bandit (exploit-only) across all four evaluation scenarios. Box plots show median, IQR, and individual samples. The bandit consistently produces tighter distributions, reducing standard deviation by up to 76%.

TACCL-generated plans. MCCS [13] addresses multi-tenant collective scheduling in cloud clusters; our work focuses on per-tenant algorithm selection.

NCCL characterization. Hu et al. [5] provide a detailed analysis of NCCL's internal mechanisms, including CTA (cooperative thread array) counts, channel configurations, and protocol behavior, but evaluate only isolated collectives without concurrent compute. Lee et al. [6] characterize the performance impact of compute-communication overlap but do not propose a tuning mechanism. Our work bridges these two directions: we measure how overlap changes the *optimal* algorithm-protocol selection (extending Hu et al.'s characterization) and build an online system that adapts to it (addressing the gap Lee et al. identify).

Bandit-based systems tuning. Multi-armed bandits [1] have been widely applied to online configuration tuning in systems. Our application to NCCL collective selection is, to our knowledge, the first use of bandits for runtime communication algorithm tuning in distributed deep learning. The key challenge we address, deterministic multi-rank agreement without communication, is specific to the collective communication setting.

7 Discussion and Limitations

Our evaluation has several limitations that scope our claims. First, we test only fp32 AllReduce; other collectives (AllGather, ReduceScatter) and data types (fp16, bf16) may exhibit different sensitivity to algorithm-protocol selection. Second, our overlap workload uses a fixed-size matmul as a proxy for backward-pass compute; real training pipelines produce variable compute loads across layers, which may shift optimal selections dynamically within a single iteration. Third, all experiments use world size 8 on A100 GPUs with NVLink and TCP interconnects; larger clusters, different GPU generations (H100, B200), or RDMA fabrics may change the relative

performance of configurations. Finally, the bandit selects a single configuration per message-size band rather than per-layer, leaving potential for finer-grained tuning.

Despite these limitations, our results demonstrate that even a simple bandit with four arms and deterministic exploration can recover 25–42% of the performance that AUTO leaves on the table at multi-node scale. The approach is general: any collective library with a pluggable algorithm-selection interface could benefit from online bandit tuning.

8 Conclusion

We have shown that NCCL's AUTO cost model, while effective on single-node NVLink topologies, leaves significant performance on the table under compute overlap and at multi-node scale. The gap amplifies from 1.2% on a single node to 57% on two nodes at 64 MB, driven by the cost model's inability to account for SM contention and heterogeneous link hierarchies.

Our online bandit tuner addresses this through a simple but effective design: deterministic round-robin exploration that guarantees multi-rank agreement, IQR-trimmed arm selection that resists outliers, and a safety gate that prevents regressions. The result is 25–42% latency reductions on multi-node topologies with zero regressions, converging in under 10 seconds.

Future work includes extending the bandit to other collectives (AllGather, ReduceScatter), testing on H100/B200 architectures with NVSwitch, integrating with production training pipelines, and exploring contextual bandits that condition arm selection on runtime compute intensity. Given NCCL's opacity in configuration selection, future work could also instrument NCCL across diverse GPU architectures, topologies, and workloads to identify patterns in its decision-making, informing tighter bandit search spaces and more interpretable arm selection.

Acknowledgments

We thank Jonathan Aimuyo and the CS244C course staff for their guidance. Experiments were run on Modal cloud infrastructure using NVIDIA A100 GPUs. Claude Opus 4.6 was used to assist with debugging Modal infrastructure issues, editing for conciseness, and figure-generation scripting.

References

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47, 2–3 (2002), 235–256. doi:10.1023/A:1013689704352

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 33.

[3] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. 2023. MSCCLang: Microsoft Collective Communication Language. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '23)*. Association for Computing Machinery, 502–514. doi:10.1145/3575693.3575724

[4] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* (2017).

[5] Zhiyi Hu, Siyuan Shen, Tommaso Bonato, Sylvain Jeaugey, Cedell Alexander, Eric Spada, James Dinan, Jeff Hammond, and Torsten Hoefler. 2025. Demystifying NCCL: An In-depth Analysis of GPU Communication Protocols and Algorithms. *arXiv preprint arXiv:2507.04786* (2025).

[6] Seonho Lee, Jihwan Oh, Junkyum Kim, Seokjin Go, Jongse Park, and Divya Mahajan. 2025. Characterizing Compute-Communication Overlap in GPU-Accelerated Distributed Deep Learning: Performance and Power Implications. In *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 353–355. doi:10.1109/ISPASS64960.2025.00041

[7] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3005–3018. doi:10.14778/3415478.3415530

[8] Xuting Liu, Behnaz Arzani, Siva Kesava Reddy Kakarla, Liangyu Zhao, Vincent Liu, Miguel Castro, Srikanth Kandula, and Luke Marshall. 2024. Rethinking Machine Learning Collective Communication as a Multi-Commodity Flow Problem. In *Proceedings of the ACM SIGCOMM 2024 Conference*. Association for Computing Machinery, 16–37. doi:10.1145/3651890.3672249

[9] NVIDIA Corporation. 2024. NCCL – External Tuner Plugin Example. GitHub repository. https://github.com/NVIDIA/nccl/tree/master/ext-tuner/example Accessed: 2026-03-11.

[10] NVIDIA Corporation. 2025. NVIDIA Collective Communication Library (NCCL) User Guide. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/overview.html. Accessed: 2026-03-09.

[11] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, 593–612. https://www.usenix.org/conference/nsdi23/presentation/shah

[12] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[13] Yongji Wu, Yechen Xu, Jingrong Chen, Zhaodong Wang, Ying Zhang, Matthew Lentz, and Danyang Zhuo. 2024. MCCS: A Service-based Approach to Collective Communication for Multi-Tenant Cloud. In *Proceedings of the ACM SIGCOMM 2024 Conference*. Association for Computing Machinery, 679–690. doi:10.1145/3651890.3672252

[14] Yang You, Aydin Buluç, and James Demmel. 2017. Scaling Deep Learning on GPU and Knights Landing Clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*. doi:10.1145/3126908.3126912