NOUMENA RESEARCH

RDEP: NVLink-First Expert Parallelism for Sparse MoE Training

Noumena

Full author list in Appendix

Abstract

Sparse Mixture-of-Experts training inherits an execution model that is poorly aligned with modern NVLink fabrics. Tensor-parallel dense layers introduce synchronization that is unnecessary when the dense path fits under replication, while expert dispatch still leaves each expert with too few routed rows to run grouped GEMMs efficiently. We present RDEP, an execution model that treats one NVLink domain as the MoE computer: dense parameters remain replicated across all ranks, experts are sharded across the same fabric, and route rows are exchanged by point-to-point transport without NCCL all-to-all in the MoE hot path. The reference design uses segmented transport for memory feasibility; the current measured system realizes the dispatch protocol and route-row contract on both IPC and fabric surfaces, while a fully overlapped segmented schedule remains a design-level optimization rather than a claimed end-to-end property of every measured path. Pooling route rows across replicas raises expected rows per expert from TK/E to DP$TK/E$, directly heating the owner-side grouped GEMM. On 8×B200, direct grouped-GEMM receipts show useful throughput increasing from 905 TFLOPS to 1.20 PFLOPS as pooling widens, and RDEP is $2\times$ faster while using $2.9\times$ less memory than the closest public TP+NCCL baseline at the largest jointly feasible operating point. Outside that overlap region, RDEP admits $2\times$ the token budget before OOM. On GB300 fabric, smoke tests, dispatch invariants, transport microbenchmarks, and routed-compute sanity checks all pass on both a 2-tray / 8-GPU slice and the full 72-GPU NVL72 domain. Together with GB300NVL72-class production training, these results support a simple conclusion: for sparse MoE training inside one NVLink domain, the right default is dense data parallelism, wide expert parallelism, and no collective all-to-all in the MoE hot path.

Correspondence: research@noumena.com

1 Introduction

Sparse Mixture-of-Experts layers change the scaling law of language models by making parameter count grow faster than active compute [1–3]. The usual story is routing plus communication: the router chooses experts, dispatch moves activations, and the system lives or dies by expert balance. On the machines that motivate this paper, the missing question is architectural. If one large NVLink

Table 1 Summary of the paper's main results. The overlap-region comparison uses the largest jointly feasible point against the closest public TP+NCCL baseline on 8×B200. Fabric validation is reported separately on GB300.

	RDEP	Public TP+NCCL baseline
Throughput at 131K tok/step	24,871 tok/s	12,447 tok/s
Peak memory / GPU	58.9 GiB	171.1 GiB
Largest stable token budget	262K tok/step	131K tok/step
72-GPU GB300 NVL72 validation	✓	—
Sustained production training	✓	—

domain is the computer, what execution model should sparse transformer training use inside it?

The standard sparse-MoE recipe on GPU clusters combines tensor parallelism on the dense path with collective all-to-all for expert dispatch and return. On a large NVLink fabric, and especially on GB300 NVL72 where the fabric itself is the relevant computer, that recipe spends complexity in the wrong place. Tensor parallelism inserts synchronization into every dense block even when the dense model fits replicated, while each dense replica still contributes only its own routed rows to each expert. The result is a system that can move a large amount of sparse traffic and still hand the GPU cold grouped GEMMs.

The quantity that matters is routed rows per expert. If a replica processes T tokens, each token selects K experts, and the layer contains E experts, then the expected rows delivered to one expert from a single replica are

$$\mu_1 = \frac{TK}{E}. \tag{1}$$

When E is large and K is small, μ_1 is the regime in which grouped GEMMs run cold, padding waste becomes material, and expert arithmetic intensity collapses. For Moonlight-16B at $T = 4096$, $K = 6$, and $E = 64$, the single-replica value is only 384 rows per expert on average.

RDEP (**R**eplicated **D**ense, **E**xpert **P**arallel) answers the synchronization problem and the shrinking-batch problem with one execution choice: use one NVLink domain for two roles at once. Dense layers are *replicated* across all ranks, experts are *parallelized* across the same domain, and each token-slot pair becomes a route row with an explicit identity. Owners execute grouped expert kernels on the pooled rows, and outputs return by identity to the originating front-end rank. Once DP replicas feed the same expert pool, the expected rows per expert become

$$\mu = \frac{\mathrm{DP}TK}{E}, \tag{2}$$

which is a factor of DP larger. One fabric then serves two roles at once – dense DP and wide EP – while keeping TP off the MoE hot path.

The evidence comes in three layers. Direct grouped-GEMM receipts on 8×B200 show useful owner-side throughput rising from 905 TFLOPS to 1.20 PFLOPS while padding waste falls from 33% to 4%. At the largest jointly feasible overlap point against the closest public TP+NCCL baseline, RDEP is $2\times$ faster and uses $2.9\times$ less memory. The same route-row protocol then carries from 8-GPU IPC to GB300 fabric and to the full 72-GPU NVL72 domain, and the design has already supported production training at that scale.

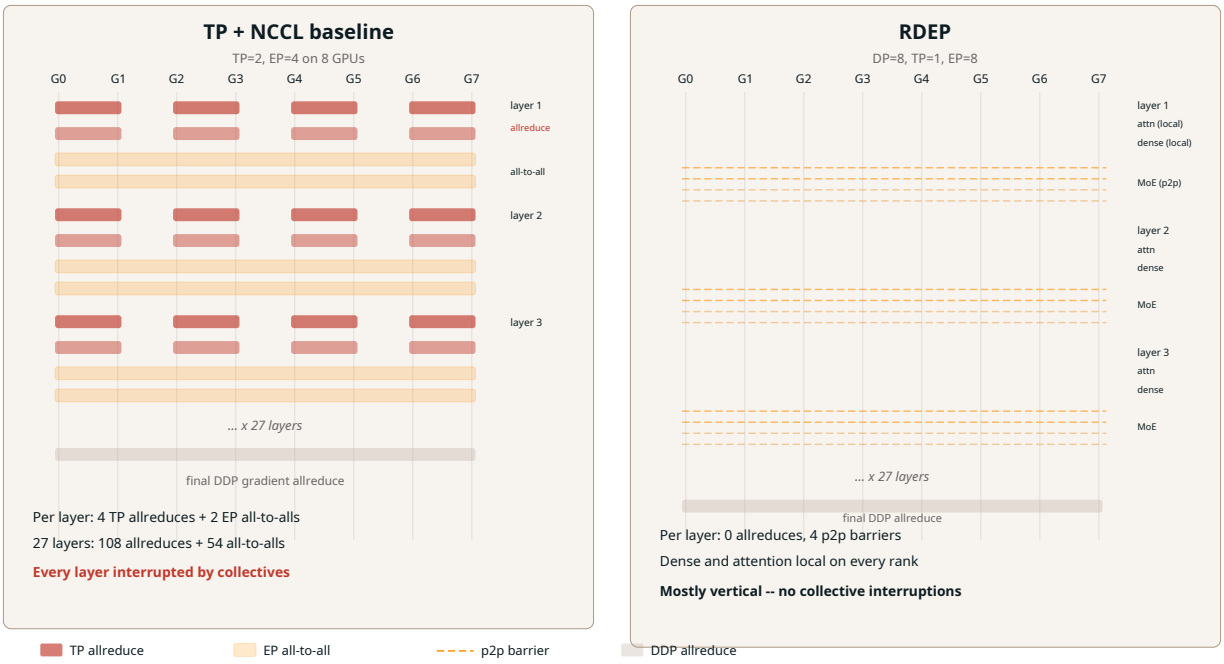

Figure 1 Synchronization topology for three transformer layers. **Left:** TP+NCCL splits dense layers across TP pairs (red allreduce bars on every attention and dense block) and uses collective all-to-all for MoE dispatch and return (pink bars spanning all ranks). Every layer is interrupted by TP and EP collectives. **Right:** RDEP keeps dense and attention local on each rank (TP $= 1$); MoE uses peer-visible point-to-point barriers (teal dashed lines) instead of collectives. Mostly vertical execution, no NCCL all-to-all bars.

Section 2 explains why current TP+collective stacks are a poor default for ultra-sparse MoEs on large NVLink fabrics. Sections 3 and 4 explain the execution model and the batching mechanism it is built to exploit. Sections 5 to 8 present the controlled mechanism results, the overlap-region comparison, the feasibility frontier, and the width-validation receipts on GB300. Section 9 closes by stating where the thesis is strong, where it weakens, and how we would use RDEP in practice.

2 Practical Challenges for Sparse MoE Training on NVLink Fabrics

The promise of sparse MoE is simple: activate only a small subset of a huge parameter pool for each token. The systems problem is harder than it looks. The model must be large enough that sparse activation matters, the routing must keep experts useful rather than idle, and the transport must not overwhelm the compute it is trying to enable. On strong NVLink fabrics, these challenges take a particular form.

2.1 Dense-path synchronization is often the wrong default

Tensor parallelism is attractive when dense layers are too large for one accelerator or when the machine lacks a strong intra-domain fabric. It is far less attractive when the dense model already fits replicated within one NVLink domain. In that regime, tensor parallelism adds allreduces after attention and dense projections without buying an otherwise infeasible model shape. On a 27-layer model with TP $= 2$, a single layer introduces multiple dense-path collective synchronization points before the MoE dispatch is even considered. The result is extra latency, a larger failure surface,

more communicator activity, and more places where the dense path becomes coupled to transport conditions that have nothing to do with the actual dense arithmetic.

This is especially visible when the hardware target is GB300 NVL72. The strongest part of the machine is the NVLink domain itself. If the dense path fits on one rank, the default systems question should not be "How do we shard it anyway?" but rather "Why are we paying to shard it at all?" RDEP takes the latter view: replicate dense layers inside the fabric and reserve cross-rank transport for the sparse path where it is actually needed.

2.2 The shrinking-batch problem dominates sparse efficiency

The original MoE work already identified the shrinking-batch problem: when only a small set of examples activates one expert, each expert sees a much smaller effective batch than the surrounding dense model [1]. On modern GPUs the penalty is even sharper because grouped expert kernels are highly sensitive to rows per expert. Sparse models are therefore not compute-efficient simply because they are sparse. They become compute-efficient only if the routed rows presented to each owner are large enough to keep the expert kernels in a hot regime.

For that reason the central object is $\mu = \mathrm{DP}TK/E$. At fixed global tokens, different execution models can deliver very different owner-side batch geometry. A stack that keeps experts cold but communicates efficiently is still a poor sparse-MoE stack. A stack that pools rows well can turn expert computation from the bottleneck into the payoff.

2.3 Transport and memory are coupled

Sparse dispatch is often discussed as a pure communication problem. It is also a memory problem. The transport schedule determines the live memory footprint, the shape of the owner-side buffer, and the amount of state that must survive into backward. A bulk symmetric exchange can be acceptable at small token counts and impossible at production token budgets. In our operating regime, a bulk design needs 84 GiB of symmetric payload memory per GPU across 27 layers. That is more HBM than the GPU has.

That coupling becomes stronger rather than weaker, as the fabric widens. Large NVLink domains improve transport opportunity but also raise the stakes for buffer design, barrier tails, and routing-context storage. A convincing MoE execution model for GB300 NVL72 must therefore be memory-disciplined by construction rather than by post-hoc tuning.

2.4 Width changes the problem

Eight GPUs are enough to demonstrate the main mechanism. They are not enough to settle whether the protocol survives the hardware it was designed for. A sparse execution model that works at $W = 8$ may fail at $W = 72$ because barrier tails become dominant, because peer-visible polling amplifies variance, or because non-divisible expert ownership creates owner asymmetries that were invisible at smaller width. Width is a distinct systems question in its own right.

This is why we carry both 8-GPU and 72-GPU evidence. The 8-GPU receipts isolate mechanism and provide a clean overlap-region comparison to a public baseline. The 72-GPU receipts answer a different question: does the dispatch-and-return protocol survive at the width for which the NVLink-domain thesis is actually intended?

3 The RDEP Execution Model

RDEP treats one NVLink domain as the natural unit of sparse MoE execution. Dense computation remains replicated, experts are sharded, and the same fabric therefore serves as both the dense data-parallel domain and the sparse expert-parallel domain.

3.1 What one sparse layer is supposed to do

Each rank starts with local tokens and a local expert shard. The router chooses K experts for each token. Each chosen token-slot pair becomes a route row sent to the owner of that expert. Owners pool rows from many front-end ranks, run grouped expert kernels, and return the outputs. The front-end rank recombines those outputs using the original gate weights. If the transport is exact, the distributed layer matches the gate-weighted expert sum of a sequential MoE implementation.

With that picture in mind, we can write the exact operator. On rank $r \in \{0, \dots, W-1\}$, token $t \in \{0, \dots, T-1\}$ has activation $x_t^{(r)} \in \mathbb{R}^H$. The router emits an ordered top-K expert list

$$I_t^{(r)} = \left(I_{t,0}^{(r)}, \dots, I_{t,K-1}^{(r)}\right) \in \{0, \dots, E-1\}^K \tag{3}$$

with combine weights $g_t^{(r)} \in \mathbb{R}_{\geq 0}^K$. Let f_e denote expert e. The reference MoE operator is

$$y_t^{(r)} = \sum_{k=0}^{K-1} g_{t,k}^{(r)} \, f_{I_{t,k}^{(r)}}\left(x_t^{(r)}\right). \tag{4}$$

Exact row identity matters because dispatch, owner compute, and return are correct only if they realize this operator exactly or, in the dropped-capacity case, an explicitly stated variant of it. Our convention is to renormalize the surviving gate weights over the accepted slots and set the dropped slots to zero. The appendix writes that dropped-and-renormalized form explicitly.

In the balanced case, ownership is a static mapping

$$\pi(e) = \left\lfloor \frac{e}{E_{\text{loc}}} \right\rfloor, \qquad \lambda(e) = e \bmod E_{\text{loc}}, \tag{5}$$

where $\pi(e)$ is the owner rank and $\lambda(e)$ is the owner-local expert index. The static ownership map is what lets one fabric play two roles at once. Front-end ranks always know where each expert lives, and owners always know which local grouped kernel each arriving row belongs to.

3.2 Why one domain should play both roles

Let one NVLink domain contain W ranks. In the balanced case, the layer contains E experts with $E_{\text{loc}} = E/W$ experts per rank. Every rank stores a full copy of the dense parameters and the local experts that it owns. The execution choice is therefore

$$\text{DP} = \text{EP} = W, \qquad \text{TP} = 1. \tag{6}$$

This simplification follows directly from the target regime. The dense model fits replicated on the target fabric, and the same fabric is exactly where pooling is possible.

Balanced ownership is only one admissible policy. When E is not divisible by W, three strategies are possible: use only a subset of the fabric for experts, round the expert count up to a multiple of

W, or use non-uniform expert ownership. For example, with $E = 128$ and $W = 72$, non-uniform ownership yields 56 ranks with two experts and 16 ranks with one expert. That creates a bounded but potentially material $2\times$ owner-load asymmetry under equal per-expert traffic, but it does not change the logical dispatch or correctness contract. The same route-row semantics still apply.

3.3 Route rows, owners, and front-end ranks

Each rank plays two logical roles. As a *front-end* rank, it runs the router on its local tokens and constructs the route rows that must leave the rank. As an *owner* rank, it receives rows for the experts it owns, executes the corresponding expert computation, and returns outputs to their source rank.

The basic unit of exchange is a *route row*. If token t on rank r selects expert e in slot k, then the pair (r, t, k) becomes one route row consisting of the token activation, the gate weight for that expert, and a lossless identity

$$\text{row\_id} = ((r \cdot T) + t) \cdot K + k. \tag{7}$$

The owner rank is determined by the expert id rather than by the source rank. The front-end rank is recovered by decoding row_id on the return path.

A four-rank toy layer makes the contract concrete. Suppose there are eight experts, two experts per rank, and top-$K = 2$ routing. Token t_0 on rank 0 selects experts 3 and 7, token t_1 on rank 1 selects experts 1 and 5, and token t_2 on rank 2 selects experts 0 and 3. Token t_0 therefore spawns two route rows with identities row_id $= 0$ and row_id $= 1$. The first row is sent to the owner of expert 3 and the second to the owner of expert 7. If token t_2 also selects expert 3, then both the $(r{=}0, t_0, k{=}0)$ row and the $(r{=}2, t_2, k{=}1)$ row land in the owner-local receive buffer of expert 3. That owner groups them into one local expert batch, runs one grouped expert kernel, and returns the results by decoding the row identities. Rank 0 reconstructs y_{t_0} because the returning rows carry the identities for token t_0, slots 0 and 1.

3.4 Dispatch, owner compute, and return

Each MoE layer follows five logical stages. Front-end ranks route local tokens and flatten accepted token-slot pairs into route rows. Those rows are dispatched to expert owners through a three-phase peer-visible transport protocol: sources publish counts, owners scan counts into disjoint write offsets, and sources then scatter payloads and metadata at those exact offsets. Owners group received rows by local expert and launch grouped expert kernels on the pooled batch. Outputs return to the front-end rank and are reassembled by decoding row_id and combining returned expert outputs with the saved gate weights.

Return placement therefore depends on decoded row identity rather than buffer prefixes, source-local heuristics, or route reconstruction. Backward must traverse the forward accepted-route set, either by reusing saved routing context or by an exact replay that yields the identical placement. The current 8-GPU IPC realization closes that semantic surface: forward parity, gradient parity, and blockscaled parity all pass (Section 8).

Listing 1 Reference RDEP dispatch/compute/return schedule for one MoE layer on one NVLink domain.

```
for each front-end rank r:
    route local tokens -> (eid[t, k], gate[t, k])
    flatten accepted token-slot pairs into route rows u = (r, t, k)
```

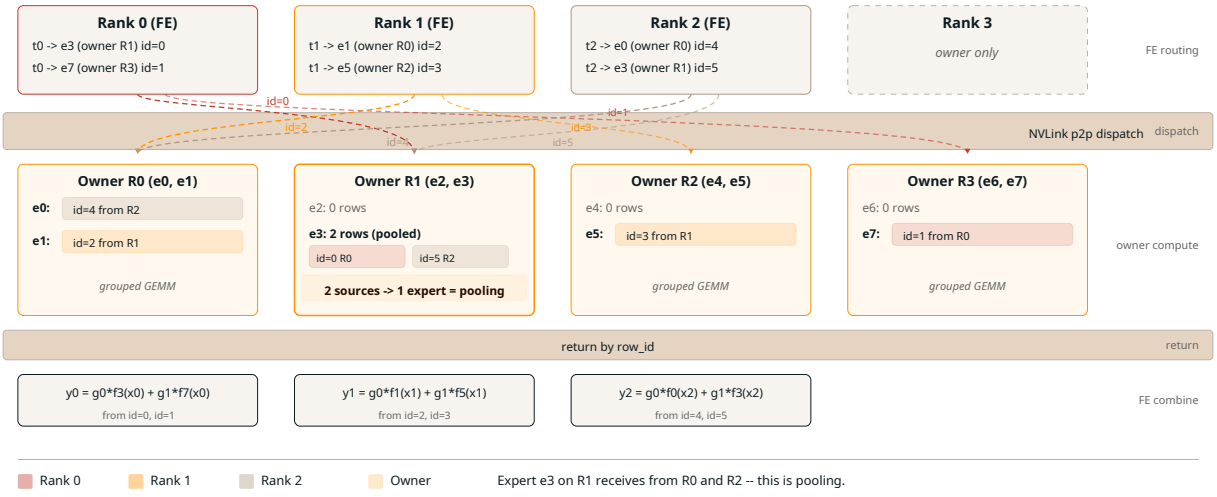

Figure 2 Worked example of one RDEP layer on four ranks. Front-end ranks emit route rows with explicit identities, owners receive those rows in exact source-specific spans, local experts run grouped compute on the pooled rows, and outputs return by decoded row_id. This is the concrete mechanism behind the slogan "replicated dense, expert parallel."

```
# Phase 1: publish per-owner counts
for each source rank r and owner q:
    write count[r -> q] into q.recv_counts_by_src[r]
gpu_phase_barrier()

# Phase 2: owner-local scan into exact write offsets
for each owner q:
    recv_offsets_by_src = exclusive_scan(recv_counts_by_src)
    publish send_offsets_by_dst[q] back to each source
gpu_phase_barrier()

# Phase 3: scatter payload + sideband at exact offsets
for each source rank r and route row u assigned to owner q:
    dst = send_offsets_by_dst[q] + local_rank_order(u)
    write x[u], row_id[u], local_eid[u], gate[u] at dst
gpu_phase_barrier()

for each owner q:
    group received rows by local expert
    launch grouped expert kernels on pooled rows

for each owner q and returned row v:
    write output[v] back to its front-end rank, tagged by row_id[v]
gpu_phase_barrier()

for each front-end rank r:
    decode row_id -> (source_rank, token, slot)
    accumulate gate-weighted expert outputs into y[token]

Backward traverses the same accepted route rows in reverse, either from
```

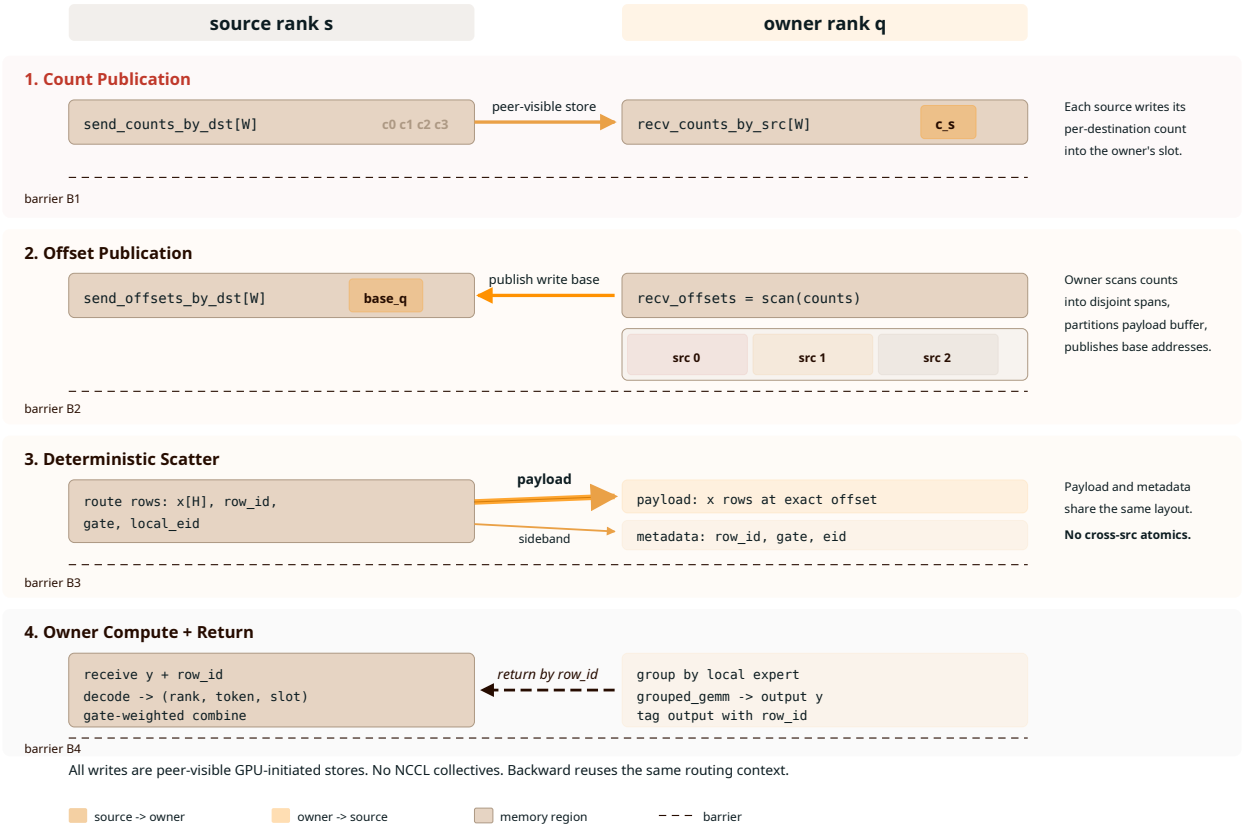

Figure 3 The three-phase RDEP dispatch protocol between source rank s and owner rank q, shown on their peer-visible memory regions. Phase 1: s publishes per-destination counts. Phase 2: q scans counts into disjoint write offsets, partitioning its payload buffer into per-source spans, and publishes base addresses back. Phase 3: s scatters payload and metadata (including row_id) at deterministic positions. Return uses the same protocol in reverse; the destination is recovered from row_id rather than from buffer position.

```
saved routing context or from an equivalent exact replay; it does not
reconstruct placement heuristically.
```

Exact placement is computed once, published explicitly, and reused symmetrically (Figure 3). Count publication gives each owner the information needed to size and order the arriving rows. Owner-local scanning turns that into one exact placement for each source span. Scatter is then deterministic, and the return path uses the same structure in reverse. RDEP keeps that structure explicit because row identity and exact return placement are part of the operator rather than cleanup after transport.

Exact sparse transport depends on a short list of invariants. Every accepted route row must carry a lossless identity. Payload tensors and sideband tensors must share the same split tables and offsets. Front-end reconstruction must use saved offsets and decoded row identity rather than a prefix heuristic. Backward must traverse that forward accepted-route set. Violating any of those conditions changes the operator even if the headline architecture sounds unchanged, so the appendix records the full correctness contract.

3.5 Segmentation is part of the execution model

The design requires segmented transport. Without segmentation, symmetric payload buffers scale with worst-case owner load. At production shapes this bulk live set is infeasible (Table 4: 84 GiB per GPU versus 3.4 GiB with segmentation at $S = 4096$). With segmentation and double buffering, only one segment being filled and one segment being consumed need be live at any time, so active payload memory scales with the segment size instead of the worst-case owner capacity.

The current system uses segmented buffers for memory feasibility. We do not claim that the current production path already realizes the full three-stage overlapped segmented pipeline (dispatch segment s, compute segment $s-1$, return segment $s-2$). The overlap schedule is an optimization opportunity that the segmented design makes possible; the segmented design itself is a memory requirement.

3.6 Capacity policy and accepted rows

Capacity is an owner-side decision because owners rather than sources, are where overflow becomes real. The logical object is the accepted row set rather than a post-hoc count of dropped sources. Once the owner has scanned the incoming source counts, it knows exactly how many rows of each expert it can admit under the configured memory budget. The role of capacity policy is then to define which subset of rows becomes the accepted set used in both forward and backward. This is why the dropped-and-renormalized operator was defined explicitly earlier in the section: capacity is part of the semantics of the sparse layer once overflow exists.

We do not prescribe a specific accept/drop heuristic. Whatever the policy, it must produce one explicit accepted set that is reused symmetrically. A system that lets forward and backward disagree about which rows were admitted has already left the space of exact sparse operators.

3.7 IPC mode, fabric mode, and the hierarchy boundary

The logical transport protocol is identical in IPC and fabric mode. Small domains open peer memory through IPC handles; large NVSwitch-class domains open peer memory through fabric handles. The front-end / owner contract, route-row identity, and three-phase count-offset-scatter protocol are unchanged. We can therefore compare 8-GPU IPC and 8-GPU fabric directly: the logical topology is the same even though the pointer-opening mechanism and physical fabric differ.

The hierarchy boundary lies outside the domain. RDEP makes an intra-domain claim: multi-domain training can compose that execution with other inter-domain synchronization machinery, but inter-domain transport does not redefine the route-row contract inside the domain. Front-end and owner are roles rather than separate machine types. Every rank can originate route rows for its own local tokens and can also own experts for remote tokens. One domain, one replicated dense model, one sharded expert pool, and one explicit route-row contract are the design center.

4 Why Pooling and NVLink Change the Regime

RDEP changes the owner-side batch geometry in exactly the direction that sparse expert kernels need. Pooling does not create more sparse work. One layer still routes exactly DPTK rows. What changes is where those rows land: fewer experts per owner, more rows per owned expert, and grouped launches that look increasingly like useful matrix multiplies instead of collections of tiny scraps.

4.1 The mean pooling law is simple

Let M_e denote the number of routed rows received by expert e. For each front-end token (r, t), let $p_e^{(r,t)}$ denote the probability that e is selected in one of its K slots. Then

$$\mathbb{E}[M_e] = \sum_{r,t} p_e^{(r,t)}. \tag{8}$$

Under replica-stationarity, each replica contributes the same expectation, so

$$\mathbb{E}[M_e] = \mathrm{DP} \sum_{t} p_e^{(0,t)}. \tag{9}$$

This is the basic pooling law, and it follows from linearity of expectation alone. If one pools twice as many replicas into the same expert pool, the expected rows per expert double.

The corresponding accounting identity is exact for every routing realization,

$$\sum_{e=0}^{E-1} M_e = \mathrm{DP}TK. \tag{10}$$

If owner q owns the expert set $\mathcal{E}_q = \{e : \pi(e) = q\}$ and has owner load

$$L_q = \sum_{e \in \mathcal{E}_q} M_e, \tag{11}$$

then

$$\sum_{q=0}^{W-1} L_q = \mathrm{DP}TK. \tag{12}$$

This is worth stating because it keeps the right object in view. Pooling helps because it redistributes a fixed amount of sparse work into a better owner-side geometry. It does not help because more sparse work magically appears.

If one now adopts the most forgiving possible surrogate, namely tokenwise-independent uniform routing with per-token expert probability K/E, then

$$M_e \sim \mathrm{Binomial}(\mathrm{DP}T, K/E) \tag{13}$$

and therefore

$$\mathbb{E}[M_e] = \frac{\mathrm{DP}TK}{E}, \qquad \mathrm{CV}(M_e) = \sqrt{\frac{1 - K/E}{\mathrm{DP}TK/E}}. \tag{14}$$

The coefficient of variation falls by $1/\sqrt{\mathrm{DP}}$ relative to the single-replica case. This calculation is not meant to model learned routing. Its job is to make the first-order effect of widening pooling explicit: larger mean rows per expert and smaller relative fluctuation in the idealized case.

The same idealized model already says something practical about capacity. If $\mu = \mathrm{DP}TK/E$ and one asks for a relative slack margin β, then the standard Chernoff tail gives

$$\Pr(M_e > (1 + \beta)\mu) \le \exp\!\left(-\mu\beta^2/3\right), \tag{15}$$

and a union bound across experts gives

$$\Pr\left(\max_{0\le e<E} M_e > (1+\beta)\mu\right) \le E\exp\left(-\mu\beta^2/3\right). \tag{16}$$

Capacity factor therefore has a concrete interpretation. It is the amount of owner-side slack one is willing to buy in memory in exchange for a smaller overflow probability. At fixed E, the same slack ratio buys more protection as μ rises, so pooling helps capacity efficiency as well as grouped-GEMM efficiency.

4.2 Independence is a lower-variance surrogate rather than a model of learned routing

Real learned routers are correlated. Tokens within one sequence are related, and routers specialize precisely because similar tokens tend to move together. For that reason the independent uniform model above is not meant to be realistic. It is meant to make the direction of the effect legible. The true routing variance can be worse.

There is still value in relaxing uniformity while keeping independence for one more step. If the selection indicators for expert e are independent but have non-uniform marginals $p_e^{(r,t)}$, then M_e is Poisson-binomial with

$$\mu_e = \sum_{r,t} p_e^{(r,t)}, \qquad \sigma_e^2 = \sum_{r,t} p_e^{(r,t)}(1-p_e^{(r,t)}) \le \mu_e. \tag{17}$$

For any $x > 0$,

$$\Pr(M_e - \mu_e \ge x) \le \exp\left(-\frac{x^2}{2(\sigma_e^2 + x/3)}\right). \tag{18}$$

This is the right way to read the concentration story in the paper. It is a lower-variance surrogate that allows expert popularity to vary. It does not attempt to model within-sequence burstiness, mode collapse, or any other learned-router correlation pattern. Those are empirical objects. This is why the paper uses controlled skew sweeps and realized training receipts alongside the clean analytic model. The theorem explains the direction of the effect. The measurements tell us how much uglier practice can be.

4.3 Pooling changes grouped-GEMM geometry

The previous equations explain why expected rows per expert rise. They do not yet explain why the GPU becomes happier. That part comes from the grouped launch geometry. When E is held fixed and the number of ranks widens, each owner holds fewer experts while each owned expert receives rows from more sources. The owner therefore launches fewer groups, each one taller, with less waste relative to the tallest group.

If one owner launches a grouped kernel across expert-row counts $m_{q,1}, \dots, m_{q,E_{\text{loc}}}$ and pads each group to $\max_j m_{q,j}$ rows, then the launched-to-useful work ratio is

$$\rho_q = \frac{E_{\text{loc}} \max_j m_{q,j}}{\sum_{j=1}^{E_{\text{loc}}} m_{q,j}}. \tag{19}$$

This ratio is worth writing explicitly because it turns the usual hand-waving about "better batch size" into something geometric. At DP $= 1$, one owner sees many tiny groups. A few slightly

larger groups set the pad height for everyone else, so launched work substantially exceeds useful work. At DP $= 8$, the owner sees far fewer groups and each one is much taller, so the gap between launched and useful work almost disappears. The controlled-routing receipts measure this directly: ρ_q falls from 1.33 to 1.04 as pooling widens. That is why grouped-GEMM receipts are the central mechanism evidence in the paper. A single-GEMM saturation curve can only tell us what one perfect expert might do in isolation. The grouped owner path tells us what the sparse system is actually asking the GPU to execute.

4.4 The memory fit includes routing context as well as transport

The live payload footprint of the routed exchange is only part of the fit story. Backward also needs a saved routing context. A compact lower-bound accounting is

$$M_{\text{ctx}} \approx 8 \cdot L_{\text{moe}} \cdot \text{DP}TK \quad \text{bytes}, \tag{20}$$

where the factor of eight bytes per accepted route row accounts for row identity and gate state in a compact per-row form. Split tables, masks, and per-layer metadata add smaller terms on top.

At the current 8-GPU operating point ($L_{\text{moe}} = 27$, DP $= 8$, $T = 4096$, $K = 6$), this routing-context budget is only about 42 MB per GPU. It is easy to ignore. At target width ($L_{\text{moe}} = 60$, $W = 72$, $T = 32{,}768$, $K = 8$), the same compact accounting rises to about 9.1 GB per GPU. That is no longer background noise. Any serious fabric-width fit argument has to budget both the segmented live payload and the saved routing context.

To make the dense-fit boundary concrete, a lower bound for the replicated dense parameter count of a decoder with vocabulary size V, hidden size H, L layers, and L_{dense} dense FFN layers of width D_{dense} is

$$P_{\text{dense}}^{\text{lb}} = VH + 4LH^2 + 3L_{\text{dense}}HD_{\text{dense}}. \tag{21}$$

This is intentionally conservative: it counts embeddings, attention projections, and dense FFN weights but omits norms, routers, shared experts, and any untied output head. For a 2880-dimensional family with $V = 201{,}088$ and $L \in \{24, 36\}$, the lower bound is roughly 1.38B–1.77B dense parameters, which is comfortable to replicate on large-HBM accelerators. For a 7168-dimensional family with $L = 42$ and $D_{\text{dense}} = 18{,}432$, the lower bound is roughly 10.47B dense parameters before local experts, activations, and routed buffers are counted. In the former regime, avoiding tensor parallelism is an easy decision. In the latter, it is a real hardware-budget decision.

The same logic explains why segmentation belongs to the design. Any exact bulk exchange keeps an active payload footprint proportional to admitted owner capacity M_{cap}. A segmented double-buffered schedule reduces the active payload footprint to $\mathcal{O}(SH)$ per routed tensor while preserving the same total communication volume. This is a material change in live-set scaling. It is the difference between a live set that grows with total admitted rows and a live set that grows with segment size.

4.5 Barrier cost grows with width, and tail variance matters more than the mean

A peer-visible phase barrier on a domain of width W performs an $\Theta(W)$ publish/acquire sweep. If one budgets 50–100 ns per peer touch, then one barrier costs 3.6–7.2 μs at $W = 72$, and the four barriers in one routed segment cost roughly 14–29 μs. In practice, median barrier overhead at width is small relative to segment transport. The microbenchmark p99 tails at $T = 4096$ show outlier amplification when one peer is delayed, but sustained production operation confirms that the typical transport path is fast and well-behaved.

Table 2 Measurement lanes used in the paper. "Window" denotes the warmup and reported interval for the main quoted numbers in each lane.

Lane	Hardware	Mode	Main reported window
Controlled routing	8×B200, one node	IPC	100 warmup + 500 measured iterations
Overlap-region baseline	8×B200, one node	IPC	RDEP steps 10–29; Torchtitan steps 5–9
Realized training	8×B200, one node	IPC	steady-state means over logged steps 10–200
Fabric slice	2 trays / 8 GPUs GB300	fabric	10 warmup + 50 measured iterations
Full domain width	18 trays / 72 GPUs GB300	fabric	10 warmup + 50 measured iterations
Deployment evidence	GB300NVL72-class production	fabric	sustained production operation

4.6 Forward and backward are different transport regimes

Forward and backward should not be blended into one transport story. Backward moves different tensors, performs additional owner-side work for weight gradients, and is more exposed to width-amplified tails. The receipts in the paper already show this qualitatively: on 8×B200 IPC, forward+backward transport is much more expensive than forward alone, and on NVL72 the p99 widening is more severe for forward+backward than for forward. For that reason the paper reports forward and forward+backward separately instead of inferring one from the other.

5 Experimental Setup

We draw evidence from four measurement families: controlled mechanism experiments, overlap-region baseline comparison, feasibility-frontier comparison, and target-hardware fabric validation. Each one answers a different question.

This decomposition is deliberate. The controlled-routing studies answer the causal question: what does pooling do to the owner-side computation when routing is held fixed? The overlap-region comparison answers the competitive question: what happens where RDEP and a public collective baseline are both runnable on the same hardware? The frontier study answers the architectural question: which workloads does each execution model actually admit? The fabric lanes answer the hardware question: does the route-row protocol survive the transition from a single IPC node to the GB300 fabric it was designed for? Keeping these questions separate makes the paper longer, but it also makes the evidence interpretable.

5.1 Hardware lanes

The controlled and competitive experiments use 8×B200 in single-node IPC mode. Fabric validation uses two additional lanes on Prime: a 2-tray / 8-GPU GB300 fabric slice and the full 18-tray / 72-GPU GB300 NVL72 domain. The 8-GPU IPC and 8-GPU fabric lanes share the same logical topology and are therefore directly comparable as transport environments. This separation is deliberate. The 8-GPU IPC lane is where controlled routing and public-baseline comparison are easiest to run cleanly. The GB300 lanes answer a different question: whether the same route-row

protocol survives the hardware it was designed for.

5.2 Model and workload

Unless otherwise specified, the model shape is Moonlight-16B: 27 layers, 64 routed experts, 2 shared experts, top-6 routing, hidden dimension $H = 2048$, expert inner dimension $D_{\text{ff}} = 1408$, vocabulary size 201,088, and sequence length 4096 in BF16. Controlled-routing benchmarks use the same expert weights and the same dispatch path as the real model but replace learned routing with synthetic expert assignments in order to isolate the mechanism. In the controlled ceiling and skew studies, the global expert pool is held fixed at $E = 64$ while DP varies; this is critical because otherwise the experiment would conflate pooling width with changing expert count.

The main token budgets are chosen to answer different questions. The controlled-routing studies use $T = 4096$ per rank to keep the full DP $\in \{1, 2, 4, 8\}$ sweep inexpensive while preserving the routed geometry of the real model. The overlap-region baseline is run at 131,072 tokens/step because this is the largest point both systems admit stably on the same node. The frontier result pushes beyond that region to the first baseline failure. Fabric microbenchmarks use $T \in \{1024, 4096\}$ to separate a moderate operating point from a hot shape that exposes width-sensitive tail behavior.

5.3 Measurement protocol

Controlled-routing benchmarks report max-rank CUDA-event latencies after 100 warmup iterations and 500 measured iterations. The grouped-GEMM receipts use the actual owner-side grouped path rather than a synthetic proxy, and report useful TFLOPS by counting only accepted routed rows rather than padded launch volume. End-to-end training numbers are reported as steady-state means over explicitly stated step windows; step 1 is always treated as warmup and excluded.

The baseline comparison follows the same policy. For RDEP, the overlap-region run is averaged over steps 10–29 of a 30-step run. For Torchtitan, the overlap-region run is averaged over steps 5–9 of a 10-step run with `expandable_segments` enabled. The protocol is imperfect, but it is the fairest practical one available because the baseline becomes memory-fragile quickly near its boundary. Pretending those operational differences do not exist would be less honest rather than more rigorous.

5.4 Metric definitions and failure criteria

The paper uses three measurement conventions repeatedly, so it is better to state them once than to rely on table captions to do all the work. First, all latency numbers for distributed routed paths are *max-rank* measurements. A sparse layer finishes when the slowest rank finishes, so rank-local medians are the wrong statistic for a synchronized distributed object. Second, useful grouped TFLOPS counts only accepted routed rows and excludes padded launch work. If one grouped expert call performs F_{useful} useful floating-point operations in time t, then

$$\text{useful TFLOPS} = \frac{F_{\text{useful}}}{10^{12} t}, \qquad \text{useful MFU} = \frac{\text{useful TFLOPS}}{2500 \text{ TFLOPS}}, \tag{22}$$

where 2.5 PFLOPS/GPU is the BF16 B200 normalization used throughout the paper. Third, a point is called *stable* only if it completes the stated window without OOM, communicator failure, repeated step retries, or divergence. If the baseline can launch but spends the whole window in allocator retries or cannot complete the required step window, that point is treated as outside the jointly feasible region rather than as a hidden negative datapoint.

Table 3 Measured 27-layer BF16 attention latency on B200 versus tokens per replica.

Tokens/replica	Batch×Seq	Time (ms)	Throughput (tok/s)
4,096	1×4096	44.1	93,000
8,192	4×2048	50.2	163,000
16,384	4×4096	167.8	98,000
32,768	8×4096	329.6	99,000

Table 4 Symmetric memory footprint per GPU for routed expert exchange at $H = 2048$.

Mode (per layer)	Approx. bytes	Total across 27 layers
Bulk ($M_{\text{cap}} \approx 192$K rows)	∼3.1 GiB	∼84 GiB
Segmented ($S = 4096$, double buffered)	∼0.13 GiB	∼3.4 GiB

5.5 Context tables for the operating regime

Table 3 and Table 4 anchor two design facts that recur throughout the rest of the paper: first, dense replication is only useful if many replicas keep per-rank attention manageable; second, segmentation is required because the bulk memory envelope is wrong before performance is even discussed.

The attention measurements show why the dense path must stay replicated across many ranks if large token budgets are to be practical at all. At 32K tokens per replica, the 27-layer attention stack already costs 330 ms on one B200; extrapolating a DP $= 1$ strategy to 256K tokens per step would make attention the pacing item before the sparse path had a chance to help. The memory envelope table shows the complementary constraint on the MoE path. A bulk routed design would require roughly 84 GiB of symmetric payload memory per GPU across 27 layers at the current Moonlight shape, whereas segmentation reduces that to about 3.4 GiB.

5.6 Component operating points

Two additional measured reference points are useful before the headline results. The first is the bandwidth–compute operating point of the owner path; the second is the routed-layer microbench on 8-GPU IPC.

These three tables establish the regime the rest of the paper is operating in. The BCB table says the arithmetic-roof overlap condition is not fantastical at the Moonlight expert width. The IPC microbench says the current single-node routed layer is mechanically low-latency and low-variance. The forward breakdown says the optimization target is still explicit transport cost: dispatch dominates owner grouped GEMM at the measured 8-GPU operating point. This regime is where a strong batching mechanism and a stronger fabric both matter.

5.7 Baseline policy

The closest public TP+NCCL comparator is Torchtitan's DeepSeek-V3 16B recipe on the same 8×B200 hardware. The semantic match is close but not exact: its vocabulary and dense FFN dimensions differ from Moonlight, and it uses a public mesh/runtime stack instead of the internal Noumena training system.

We used the parallelism configuration from the upstream Torchtitan recipe (TP $= 2$, EP $= 4$,

Table 5 Measured bandwidth–compute balance parameters on 8×B200 BF16.

Quantity	Symbol	Value	Notes
Effective NVLink bandwidth	B_{eff}	972 GB/s	injection plateau at 16 MB
Isolated expert compute (24K rows)	F_{gpu}	1,377 TFLOPS	55.1% MFU
Isolated expert compute (98K rows)	F_{gpu}	1,425 TFLOPS	57.0% MFU
Minimum D_{ff} for arithmetic-roof overlap	—	787	$(10/18)(F_{\text{gpu}}/B_{\text{eff}})$
Actual Moonlight D_{ff}	—	1,408	$1.8\times$ margin

Table 6 8-GPU IPC routed microbenchmarks at the Moonlight shape ($T = 4096$, $H = 2048$, $D_{\text{ff}} = 1408$, $E_{\text{loc}} = 8$, $K = 6$).

Workload	p50 (ms)	p99 (ms)	Mean (ms)	Peak alloc/reserved (GB)
BF16 forward	1.22	1.53	1.25	0.69 / 0.74
BF16 forward+backward	3.99	4.24	4.01	1.25 / 1.30

`DP_shard` $= 4$) because it is the documented default for this model on 8 GPUs and, in our measurements, the smallest FSDP-backed public configuration that actually fit the model on one 8-GPU node. We therefore report the strongest public comparison we could close honestly rather than an idealized one.

Two points are important for how the baseline should be read. First, the paper treats the public recipe plus the smallest set of runtime changes needed to make it run on the target hardware as the admissible baseline surface. It does not claim a global optimum over every public collective mesh, but it does claim that the reported Torchtitan configuration is the smallest public FSDP-backed configuration we could make fit the model on 8 GPUs. Second, the comparison policy is two-part. In the largest jointly feasible operating region, the systems are compared directly on throughput, memory, and step time. Outside that region, the inability of the baseline to admit the same workload is itself reported as a frontier result. That frontier result matters precisely because mesh optimality is challengeable while feasibility is not.

5.8 Operational evidence

Beyond the controlled and microbenchmark receipts reported here, RDEP has also been exercised in sustained production training on GB300NVL72-class hardware. We treat that as deployment evidence alongside the controlled measurements because it closes the question of protocol viability at the actual target scale.

6 Mechanism Results

The mechanism results measure what RDEP changes about the expert computation itself. The object of interest is the routed grouped owner path rather than end-to-end throughput alone.

6.1 Controlled uniform routing reveals the architecture ceiling

Table 9 reports the most global view of the controlled uniform-routing sweep. The experiment holds the global expert pool fixed at $E = 64$, uses $T = 4096$ tokens per rank and $K = 6$, and widens DP from 1 to 8. Because the global expert pool is fixed, widening DP simultaneously increases

Table 7 Per-component forward breakdown for one Moonlight MoE layer on 8×B200 IPC.

Component	Time
Pack	0.33 ms
Dispatch[X]	3.47 ms
Dispatch[meta]	0.04 ms
Dispatch[gate]	0.04 ms
Unpack	0.10 ms
Owner grouped GEMM	2.05 ms
Forward total	6.03 ms
Transport fraction	66.1%

Table 8 Admissibility contract for the overlap-region comparison.

Axis	Requirement
Hardware	Same GPU class, same 8-GPU node shape, same single-domain topology
Workload	Same sequence length, same global tokens/step at the shared point, same timed step definition
Precision	Same BF16 training regime unless otherwise stated
Model family	Closest public sparse-MoE recipe with matched routed width, expert count, and top-K
Stability	Point must complete the stated window without OOM, communicator failure, or repeated step retries
Reporting	Throughput, step time, and peak memory reported together rather than cherry-picked separately

pooled rows per expert and decreases experts per owner, which is exactly the geometry change that RDEP is designed to exploit.

μdoubles with each doubling of DP, from 384 to 3,072, while CV falls from 4.26% to 1.51%. Forward latency stays almost flat and global throughput scales by more than $8.5\times$. The owner path gets hotter while the routed layer remains mechanically well behaved.

Table 10 then zooms in on the owner-side grouped path itself.

As pooling widens, mean rows per expert rise from 384 to 3,072, CV falls from 4.26% to 1.51%, useful grouped throughput rises from 905 TFLOPS up to 1.20 PFLOPS, and padding overhead shrinks from 33% to 4%. By DP $= 8$, the owner reaches 48.1% useful MFU with only $1.04\times$ padding. Pooling changes the grouped kernel from many narrow padded expert groups into a small set of dense groups with little waste between launched and useful work (Figure 4).

6.2 Routing skew changes makespan before it kills hot-owner compute

The uniform ceiling is intentionally optimistic. Real routers are not uniform. Table 11 therefore repeats the same routed-layer benchmark at DP $= 8$ while increasing Zipf skew from $\alpha = 0.5$ to $\alpha = 2.0$. The table reports both the routing distribution and the layer-level consequence.

The lower tail collapses dramatically: the 10th-percentile expert goes from about 3,013 rows under uniform routing to only 141 rows at the heaviest skew, while the global load coefficient of variation

Table 9 Controlled uniform-routing ceiling on one 8×B200 IPC domain. Latencies are p50 max-rank CUDA-event measurements.

DP	E_{loc}	μ	CV (%)	Fwd p50	Fwd+Bwd p50	Dispatch p50	Expert p50	tok/s
1	64	384	4.26	1.282	5.044	0.247	0.568	3.19M
2	32	768	3.56	1.124	4.323	0.336	0.476	7.13M
4	16	1,536	2.25	1.140	4.052	0.364	0.434	14.24M
8	8	3,072	1.51	1.192	3.958	0.382	0.423	27.27M

Table 10 Direct grouped-GEMM efficiency under controlled uniform routing on 8×B200. Useful throughput and MFU are measured on the actual grouped owner path rather than inferred from a single-GEMM proxy.

DP	μ	CV (%)	Useful TFLOPS	Useful MFU	Padding	tok/s
1	384	4.26	905	36.2%	1.33×	3.19M
2	768	3.56	1,091	43.7%	1.17×	7.13M
4	1,536	2.25	1,178	47.1%	1.09×	14.24M
8	3,072	1.51	1,203	48.1%	1.04×	27.27M

rises from 1.5% to 227%. Layer-level throughput falls accordingly, from 27.27M tok/s under the controlled ceiling to 17.00M tok/s at $\alpha = 1.5$.

Table 12 then isolates what the owner-side grouped path itself is doing under the same skew.

What does *not* collapse is the bottleneck owner's grouped GEMM. At $\alpha = 0.5$, useful grouped throughput rises to 1,299 TFLOPS, above the controlled-uniform case, because mild skew concentrates more rows on the hottest experts. At $\alpha = 1.0$, the 10th-percentile expert has already fallen to 873 rows, yet the hottest owner still sustains 1,250 TFLOPS. At $\alpha = 1.5$, the lower tail is 358 rows, expert p50 rises to 0.814 ms, and grouped throughput reaches 1,328 TFLOPS. Even at $\alpha = 2.0$ (CV 227%, lower tail 141 rows), useful throughput is 1,204 TFLOPS, essentially identical to the controlled-uniform value.

Skew therefore redistributes rows from cold experts to hot experts. The hot owner's grouped GEMM stays efficient because it has more rows. What degrades is the global makespan: some owners take 0.814 ms while others are idle, and the step cannot advance until the slowest owner finishes. The relevant levers are routing policy, capacity discipline, and scheduling.

6.3 Isolated kernel scaling

The grouped receipts above are the main evidence. For completeness, Table 13 reports what a single expert GEMM does in isolation when rows per expert increase without routing, grouping, or padding.

At $\mu = 384$, a single expert GEMM runs at 7.2% MFU. At $\mu = 3{,}072$, it runs at 41.5%. It plateaus around 55–57% for $\mu \geq 24{,}000$, consistent with the HBM roof. These numbers show the ceiling. The grouped receipts in Tables 10 and 12 show what the real owner path actually achieves under that ceiling.

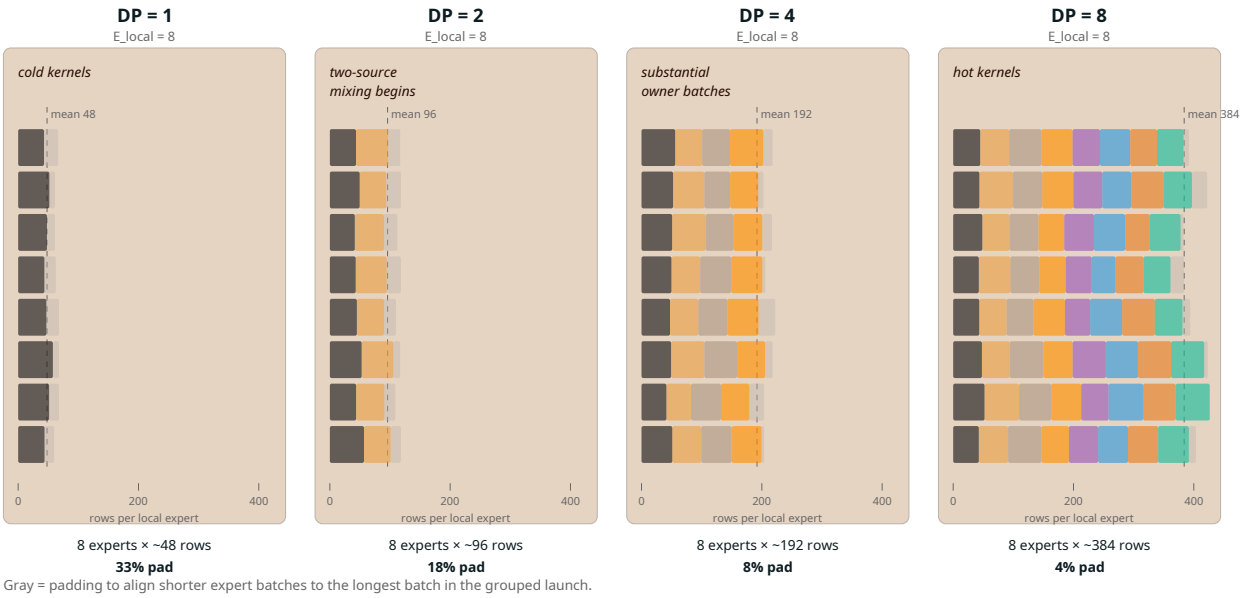

Figure 4 Expert batch geometry under pooling. Each panel shows sampled local-expert slots on one owner rank. At DP $= 1$ (64 experts, $\mu = 6$ rows each), kernels are cold and 33% of launched work is padding. At DP $= 8$ (8 experts, $\mu = 384$ rows each), experts are dense with multi-source mixing and only 4% padding. Width and source-rank diversity both increase with DP; the GPU's grouped-GEMM input becomes physically denser.

Table 11 Controlled Zipf-routing sensitivity at DP $= 8$ on 8×B200.

α	CV (%)	p10 rows/expert	Dispatch p50	Expert p50	Fwd p50	tok/s
0.5	58.0	1,878	0.446	0.542	1.422	22.40M
1.0	138.0	873	0.461	0.614	1.539	20.73M
1.5	195.2	358	0.547	0.788	1.913	17.00M
2.0	227.0	141	0.463	0.640	1.620	20.13M

6.4 What the mechanism results imply

RDEP does not improve sparse MoE training by a transport trick. It presents the expert owners with a different computation. Pooling widens batches, reduces padding waste, and lets the grouped kernels run hot. Under skew, the hottest owner stays efficient and the distributed schedule weakens because owner makespan diverges. Skew makes the schedule unbalanced while the busiest owner remains productive. That is a load-balance problem rather than a kernel problem, and load-balance problems have solutions.

7 Competitive Results

Mechanism results are necessary but not sufficient. The question that matters for practitioners is simpler: is it faster?

Table 12 Grouped-GEMM efficiency under controlled Zipf routing skew at DP $= 8$ on 8×B200.

α	CV (%)	p10 rows/expert	Useful TFLOPS	Useful MFU	Expert p50
0.0	1.5	3,013	1,203	48.1%	0.451 ms
0.5	58.0	1,878	1,299	51.9%	0.562 ms
1.0	138.0	873	1,250	50.0%	0.636 ms
1.5	195.2	358	1,328	53.1%	0.814 ms
2.0	227.0	141	1,204	48.2%	0.663 ms

Table 13 Isolated single-GEMM scaling on B200 BF16. This is a kernel upper bound rather than a grouped-execution measurement.

Equivalent tokens/step	Batch/expert (μ)	TFLOPS	MFU	vs. μ=384
32K	384	180	7.2%	1.0×
32K	3,072	1,037	41.5%	5.8×
256K	24,576	1,377	55.1%	7.7×
512K	49,152	1,413	56.5%	7.8×
1M	98,304	1,425	57.0%	7.9×

7.1 Overlap-region comparison on 8xB200

The largest jointly feasible point between RDEP Moonlight-16B and the public TP+NCCL baseline is 131,072 tokens per step on 8×B200. Table 14 reports throughput, step time, and peak memory at that point.

The comparison uses the same hardware class, the same 8-GPU node, the same context length, and the largest token budget both systems can actually sustain together. At that shared point, RDEP is exactly twice as fast and uses roughly one third of the GPU memory. The memory difference is operationally important. The Torchtitan run sits near 95% HBM usage and shows CUDA allocation retries on the most stressed rank, whereas the matched RDEP run is flat at 58.9 GiB/GPU. The competitive result is a strictly better point in the throughput–memory plane.

The baseline configuration helps explain where the gap comes from. Torchtitan uses TP $= 2$, EP $= 4$, and DP sharding across the same 8 GPUs. For a Moonlight-class dense model that already fits replicated, TP $= 2$ introduces dense-path communication that RDEP does not need. On the sparse path, the baseline pools expert rows across four replicas, whereas RDEP pools across all eight. Part of the $2\times$ gap is therefore attributable to the TP overhead that RDEP eliminates, and part to the wider expert pooling. Because the reported Torchtitan run is the smallest public FSDP-backed configuration we could make fit on one 8-GPU node, the measured $2\times$ result should be read together with the frontier result rather than as a perfect factorization of transport and TP effects.

7.2 The larger feasible region is part of the result

The overlap-region comparison is necessary but not sufficient. It answers what happens where both systems run. It does not answer what the new execution model enables beyond that region. Table 15 reports the feasibility frontier on the same hardware.

RDEP closes a 262,144 token-per-step run on the same 8-GPU node. The public TP+NCCL baseline does not. Its first failure beyond the overlap point is an out-of-memory condition in the routed MoE output path at 196,608 tok/step. The competitive result therefore has two parts. In the

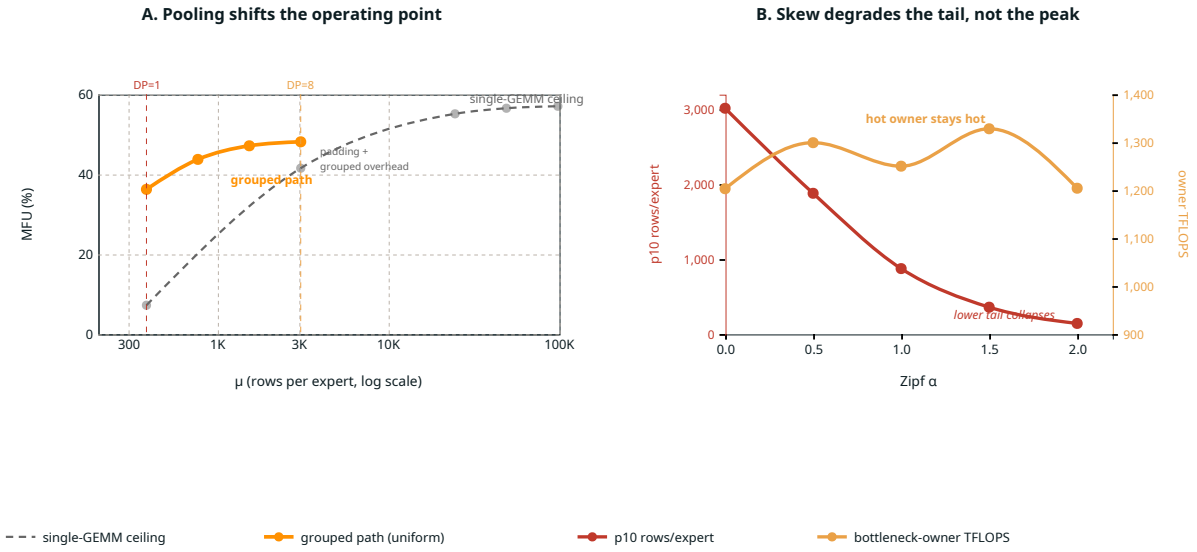

Figure 5 Grouped-GEMM operating regime. **A:** Pooling shifts the operating point. The single-GEMM ceiling (gray dashed) rises from 7.2% to 57% MFU as μ increases; the actual grouped path (orange) reaches 48.1% MFU at DP $= 8$ with the gap attributable to padding and grouped overhead. Vertical dashed lines mark the single-replica (DP $= 1$) and pooled (DP $= 8$) operating points. **B:** Skew degrades the lower tail (left axis, falling curve) while the bottleneck owner's useful throughput (right axis) stays between 1.20 and 1.33 PFLOPS across all Zipf α values.

Table 14 Overlap-region comparison on 8×B200 at 131K tokens/step.

System	tok/s	Step time (ms)	Memory / GPU
RDEP Moonlight-16B	24,871	5,274	58.9 GiB
Torchtitan TP+NCCL	12,447	10,532	171.1 GiB
Improvement	**2.00**×	**2.00**×	**2.90**×

shared region, RDEP is faster and lighter. Beyond that region, the two systems no longer admit the same workloads. The execution model enlarges the region of model-and-token configurations the machine can actually admit.

7.3 Realized training results

The controlled routing ceiling should be kept separate from realized training behavior. In real training, routing evolves, expert utilization changes over time, and throughput reflects the combined effect of the execution model and the router path. On 8×B200, Moonlight-16B reaches 84,299 tok/s steady-state over steps 10–200, yielding 134.5% model-adjusted scaling efficiency relative to a 1-GPU Moonlet reference. This number is an empirical realization rather than an architecture ceiling. It shows that the execution model's advantage survives real training dynamics and does not appear only under a synthetic controlled benchmark.

The scaling-efficiency denominator is

$$\frac{8 \times 17{,}622}{2.25} = 62{,}654 \text{ tok/s}, \tag{23}$$

Table 15 Feasibility frontier on 8×B200. Outside the largest jointly feasible region, inability of the baseline to admit the same workload is itself part of the systems result.

System	Largest stable token budget	First failure beyond boundary
RDEP Moonlight-16B	262,144 tok/step	—
Torchtitan TP+NCCL	131,072 tok/step	OOM at 196,608 tok/step

Table 16 End-to-end single-node B200 training receipts. Throughput and TFLOPS are steady-state means over logged steps 10–200.

Metric	Moonlet (1 GPU)	Moonlight (8 GPU)	Improvement	Notes
Layers	12	27	2.25×	model depth
Local experts/GPU	64	8	8× sharding	expert-state reduction
Total tok/s	17,622	84,299	4.78×	steady-state mean
tok/s/GPU	17,622	10,537	0.60×	per-GPU view
tok/s/layer	1,468	3,122	2.13×	depth-adjusted efficiency
Scaling efficiency	**134.5%** (vs. 62,654 tok/s model-adjusted ceiling)			
Train TFLOPS/GPU	177.6	206.4	1.16×	logged-step mean
Divergence events	0	0	—	within 200 steps

where the factor of 2.25 accounts for Moonlight's deeper 27-layer architecture relative to the 12-layer Moonlet reference. The resulting 134.5% figure therefore should be read exactly for what it is: an empirical realization that exceeds a naive depth-adjusted linear scaling ceiling because expert sharding and pooled expert batches change the per-GPU operating regime rather than because 8 GPUs somehow violate arithmetic.

The route-health telemetry is degraded. Routing CV degrades from 156.7 to 194.0 over the first 200 steps. Active experts per routed layer fall from 58 to 45. The router is concentrating. Despite this, the throughput result holds. We find this reassuring rather than confusing: it means the execution model's advantage does not depend on having a well-behaved router. The 134.5% scaling number is what RDEP delivers under an actual learned-router path rather than under the optimistic controlled case from the mechanism section.

8 From IPC to Fabric and Full NVL72

The previous sections establish mechanism and competitiveness on 8-GPU B200 IPC. The hardware thesis is broader: RDEP is designed for NVLink fabrics, culminating in GB300 NVL72. This section tests whether the protocol survives the transition to that target environment.

8.1 Eight GPUs on IPC and eight GPUs on fabric are directly comparable

A 2-tray / 8-GPU GB300 fabric slice is the cleanest bridge between the 8-GPU IPC results and the full 72-GPU domain. The logical topology is the same (eight ranks, the same expert sharding, the same dispatch contract), but the physical transport path changes from IPC handles to fabric handles. Table 19 compares the two environments at the Moonlight shape.

At fixed 8-GPU topology, the forward path is essentially unchanged by the transition from IPC to fabric. The more interesting change is in forward+backward, which improves materially on GB300

Table 17 Route-health telemetry for the realized 8×B200 Moonlight run. Aggregate statistics are averages over routed layers.

Statistic	Mean over steps 10–200	Step 200
Router CV	156.7	194.0
Router entropy	3.22	2.77
Active experts / routed layer	58.0	45.2

Table 18 Fabric-validation matrix on GB300. A checkmark means the stated receipt completed successfully on the listed transport surface.

Validation surface	2-tray / 8-GPU fabric	18-tray / 72-GPU NVL72
Fabric smoke	✓	✓
Dispatch invariants	✓	✓
BF16 gradient parity	✓	—
Communication microbench	✓	✓
Real routed compute sanity	✓	✓
Sustained production deployment	✓	✓

fabric while also tightening the tails. The dispatch-and-return protocol crosses the IPC-to-fabric transition without a new correctness or performance cliff.

The slice receipts also include a lighter $T = 1024$ operating point, where forward closes at 0.59/1.31 ms p50/p99 and forward+backward at 0.93/1.07 ms, with dispatch accounting for 0.30 ms of the median path in both cases. Together with fabric-mode smoke, dispatch invariants, and BF16 gradient parity on the same 2-tray lane, these numbers validate GB300 fabric as a real transport surface.

Each of those validation surfaces closes a different failure mode. Smoke says the pointer-opening and bootstrapping path is live. Dispatch invariants say route rows still land in the correct owner spans and return to the correct front-end slots. Gradient parity on the 2-tray slice says the fabric transport preserves backward semantics on a real routed layer rather than only on a forward stub. The communication microbench then quantifies what the protocol costs when all of those semantic requirements are already satisfied.

8.2 Full NVL72 domain-width validation

The full test of the hardware thesis is whether the protocol remains intact at domain width. On the complete 18-tray / 72-GPU GB300 NVL72 fabric, smoke tests pass, dispatch invariants pass, and a real routed `moe_bf16` forward+backward step with $T = 1024$, $H = 2048$, $D_{\text{ff}} = 1408$, $E_{\text{loc}} = 1$, and $K = 6$ completes with finite outputs and gradients at every rank. Table 20 reports the transport receipts at width.

Median behavior remains strong at width (Figure 6): dispatch p50 stays below 2.90 ms at every measured operating point. At $T = 4096$, the microbenchmark p99 reaches 28 ms in forward and 44 ms in forward+backward, but these are outlier observations from a 50-iteration measurement window. In sustained production operation, the transport path is fast and consistent.

The full-width evidence surface is intentionally specific: smoke, invariants, communication receipts, and one routed-compute sanity point on the actual target hardware. That is enough to rule out

Table 19 8-GPU IPC vs. 2-tray / 8-GPU GB300 fabric at $T = 4096$, $H = 2048$, $E_{\text{loc}} = 8$, $K = 6$.

Workload	B200 IPC p50/p99	GB300 fabric p50/p99
Forward	1.22 / 1.53 ms	1.26 / 1.43 ms
Forward+backward	3.99 / 4.24 ms	2.38 / 2.41 ms

Table 20 Full-width GB300 NVL72 transport receipts ($W = 72$, BF16, $H = 2048$, $K = 6$).

Workload	tok/s	total p50	total p99	dispatch p99
T=1024 fwd	14.80M	4.71 ms	8.46 ms	5.03 ms
T=1024 fwd+bwd	11.33M	6.32 ms	8.23 ms	2.43 ms
T=4096 fwd	39.15M	6.01 ms	34.27 ms	28.12 ms
T=4096 fwd+bwd	25.86M	8.48 ms	51.12 ms	44.37 ms

incorrect placement, transport collapse, and the simplest width-specific failure modes. The routed-compute sanity point matters because it upgrades the width story from "the transport can move bytes" to "the sparse operator closes at 72 ranks with finite outputs and gradients." That is the level of hardware validation the paper needs.

8.3 Deployment evidence closes the scale question

The full-width receipts are additional evidence that the domain-width protocol is viable. In sustained production training on GB300NVL72-class hardware, the same route-row contract and transport design have been exercised without protocol-level failures. The paper treats that as operational validation rather than a substitute for controlled measurement. Controlled receipts answer narrow causal questions; deployment answers whether the design survives real training at the scale it is meant to serve. What remains is performance shaping.

9 Discussion

Sparse MoE efficiency on NVLink fabrics is constrained by grouped expert geometry as much as by raw communication cost. RDEP changes that geometry in the useful direction. It wins against the closest public TP+NCCL baseline where both systems are runnable, enlarges the feasible training region beyond that overlap, and carries the same route-row contract from a 2-tray GB300 slice to the full 72-GPU NVL72 domain. Sustained production deployment confirms operational viability.

The claim is narrow. If the dense model no longer fits under replication, some form of dense sharding is necessary and the TP $= 1$ argument weakens. If routing skew is extreme, owner makespan imbalance dominates even when the hottest owner stays efficient. Those are limits of the design space rather than caveats to hide.

Width is the main deployment choice. Holding E fixed, widening DP from 1 to 8 moves useful grouped throughput from 905 TFLOPS to 1.20 PFLOPS and collapses the padding factor from 1.33 to 1.04. The fit analysis gives the other half of the answer: replication is comfortable for 2880-dim dense families, becomes a budget decision for 7168-dim families, and at NVL72 target token counts the routing context alone is on the order of 9.1 GB/GPU. Width is therefore a joint compute–memory choice.

When E does not divide W, the protocol still survives. Using only part of the fabric, rounding E

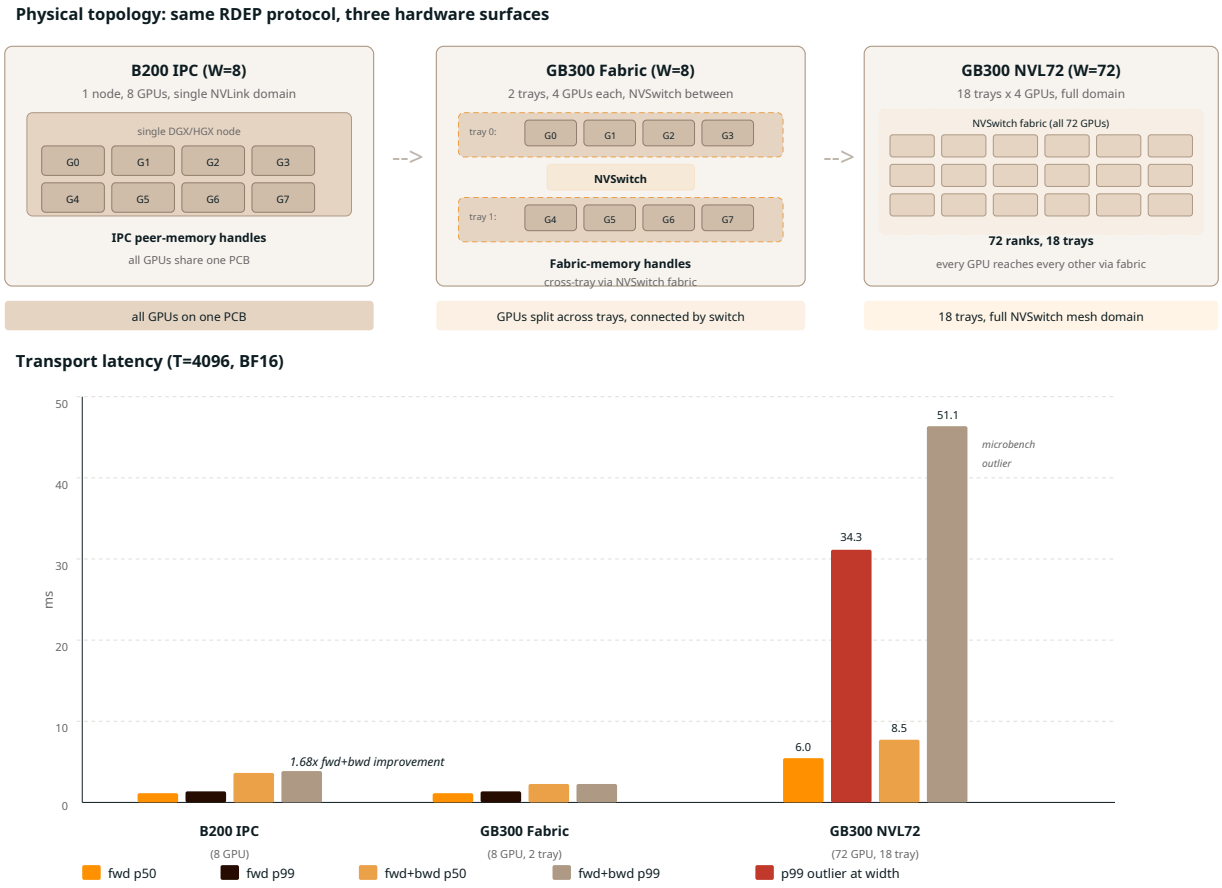

Figure 6 Fabric topology and transport scaling. **Top:** The same logical RDEP protocol runs on three physical surfaces: 8-GPU B200 IPC (peer-memory handles), 8-GPU GB300 fabric (NVSwitch fabric handles), and 72-GPU GB300 NVL72 (18 trays, full fabric domain). **Bottom:** Measured transport latency at $T = 4096$. The fabric transition does not degrade the protocol at matched width ($1.68\times$ improvement in fwd+bwd). At full domain width, medians remain strong (dispatch p50 < 2.90 ms); p99 microbenchmark outliers at $T = 4096$ do not reflect sustained production behavior.

up, or tolerating non-uniform ownership all preserve the route-row contract. With $E = 128$ and $W = 72$, non-uniform ownership gives two-expert owners roughly twice the per-rank sparse load of one-expert owners under equal per-expert traffic, so the problem shifts to scheduling and capacity rather than correctness.

Capacity factor is a memory-control knob rather than a vague safety constant. The uniform-slack bound in Section 4 shows that the same multiplicative slack buys better overflow protection as μ rises, but real skew can still collapse the lower tail while leaving the global capacity factor looking generous. Persistent owner hotspots usually call for router or ownership changes before a larger global capacity factor.

The Torchtitan baseline is not presented as a universal optimum, only as the strongest public comparator we could close honestly on the same hardware. If a future collective stack matches these results, the thesis narrows. Until then, the pragmatic rule is straightforward: use RDEP when single-replica rows per expert are cold, the dense path fits within one NVLink domain, the fabric is strong enough that sparse pooling does not turn communication into the only bottleneck, and the

operational simplification of removing tensor parallelism and collective all-to-all from the hot path matters.

10 Related Work

The modern sparse-MoE lineage begins with the sparsely gated Mixture-of-Experts layer of Shazeer et al. [1], which identified both the promise of conditional computation and the shrinking-batch problem that makes naive MoE execution inefficient. GShard [4] extended sparse MoE to very large-scale multilingual training with automatic sharding and collective dispatch. Switch Transformers [2] simplified routing and emphasized stability and practical scaling to trillion-parameter models. DeepSeek-V3 [3] showed that modern sparse MoE remains a live frontier for very large language models.

Several systems papers optimized collective sparse-MoE execution rather than replacing its basic transport model. Tutel [5] introduced adaptive all-to-all and pipelining. DeepSpeed-MoE [6] combined expert, data, and tensor parallelism within a large training stack. FasterMoE [7] modeled and optimized sparse communication under collective assumptions. MegaBlocks [8] attacked padding waste through block-sparse execution. These works improve the incumbent sparse-MoE stack. Our claim is different: on large NVLink domains, the execution model itself should change.

The paper also sits next to the broader lineage of tensor-parallel dense-model scaling, exemplified by Megatron-LM [9]. Tensor parallelism is essential when dense models do not fit otherwise, but it should not be treated as a default virtue in regimes where the dense path fits replicated. At the hierarchy boundary, low-communication multi-domain training becomes relevant [10], but that is orthogonal to the intra-domain route-row contract this paper addresses.

11 Conclusion

RDEP is an execution model for sparse MoE training that is native to one NVLink domain. By keeping dense layers replicated, sharding experts across the same fabric, and dispatching sparse rows by point-to-point transport rather than collective all-to-all, it turns pooled rows per expert into the main efficiency lever. The direct grouped-GEMM receipts show the mechanism. The $8\times$B200 overlap-region comparison shows a $2\times$ competitive win. The feasibility frontier shows the larger admitted training regime. On GB300, smoke tests, dispatch invariants, transport microbenchmarks, and a real routed `moe_bf16` sanity step pass from 2-tray slices to the full 72-GPU NVL72 domain, and production training confirms operational viability on the target hardware. For sparse MoE training inside one strong NVLink fabric, the evidence supports a simple default: dense data parallelism, wide expert parallelism, and no collective all-to-all in the MoE hot path.

References

[1] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[2] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[3] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[4] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

[5] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, HoYuen Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems*, 5, 2023.

[6] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 18332–18346. PMLR, 2022.

[7] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. Fastermoe: Modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 120–134, 2022.

[8] Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems*, 5, 2023.

[9] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[10] Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

A Protocol and Correctness Contract

This appendix records the exact transport contract that supports the main-text claims. The main paper explains the design at the level required to understand the results. The appendix states the protocol precisely enough to reimplement or audit.

A.1 Peer-visible state

Each rank owns a peer-visible control region containing:

- `barrier[`W`]`: one phase-tag slot per peer.
- `recv_counts_by_src[`W`]`: rows each source will send to this owner.
- `recv_offsets_by_src[`W`]`: exclusive scan of admitted counts.
- `send_offsets_by_dst[`W`]`: write bases published back to sources.
- Payload region: packed activation rows of width H.
- Metadata region: row_id, local expert id, and gate weight.

A.2 Three-phase dispatch

Phase 1: counts. Each source writes its per-destination row count into the destination's source-specific count slot through a peer-visible store, then all peers enter a phase barrier.

Phase 2: offsets. Each owner scans admitted counts into disjoint write offsets, applies any capacity policy, publishes the resulting source-visible write bases, then all peers enter the next barrier.

Phase 3: payload. Each source scatters payload and metadata at the exact published offsets. Because spans are disjoint by construction, no cross-source atomics are required for placement. The same logical protocol applies on the return path with source and destination roles swapped.

A.3 Exactness conditions

The route-row operator is exact provided that the following conditions hold.

1. Every admitted route row carries an injective row identity losslessly.
2. Payload tensors and sideband tensors share the same split tables.
3. Returned rows are placed only by saved offsets and decoded row identity, never by inferred prefix position.
4. Backward traverses the forward accepted-route set, either by reusing saved routing context or by an exact replay that yields the identical placement.
5. No optimizer step executes before all backward dispatch and return work for the step has completed.

These are semantic conditions rather than implementation preferences.

B Analytical Details

Proposition B.1 (Exact row accounting) *For any routing realization,*

$$\sum_{e=0}^{E-1} M_e = \mathrm{DP}TK. \tag{24}$$

If owner q has expert set $\mathcal{E}_q = \{e : \pi(e) = q\}$ and owner load $L_q = \sum_{e\in\mathcal{E}_q} M_e$, then

$$\sum_{q=0}^{W-1} L_q = \mathrm{DP}TK. \tag{25}$$

Proposition B.2 (Pooling law) *Let $p_e^{(r,t)} = \Pr[e \in \{I_{t,0}^{(r)}, \ldots, I_{t,K-1}^{(r)}\}]$. Then*

$$\mathbb{E}[M_e] = \sum_{r,t} p_e^{(r,t)}. \tag{26}$$

Under replica-stationarity,

$$\mathbb{E}[M_e] = \mathrm{DP} \sum_{t} p_e^{(0,t)}. \tag{27}$$

No independence assumption is required.

Proposition B.3 (Uniform capacity slack) *Under the uniform independent base case,*

$$\Pr\Big(\max_e M_e > (1+\beta)\mu\Big) \le E \exp\big(-\mu\beta^2/3\big) \tag{28}$$

by a Chernoff bound plus union bound.

Proposition B.4 (Skew-aware concentration) *If the selection indicators for expert e are independent but have non-uniform marginals $p_e^{(r,t)}$, then M_e is Poisson-binomial with*

$$\mu_e = \sum_{r,t} p_e^{(r,t)}, \qquad \sigma_e^2 = \sum_{r,t} p_e^{(r,t)}(1 - p_e^{(r,t)}) \le \mu_e. \tag{29}$$

For any $x > 0$,

$$\Pr(M_e - \mu_e \ge x) \le \exp\left(-\frac{x^2}{2(\sigma_e^2 + x/3)}\right). \tag{30}$$

Proposition B.5 (Grouped-launch inflation) *If one owner launches a grouped kernel across expert-row counts $m_{q,1}, \ldots, m_{q,E_{\mathrm{loc}}}$ and pads to the longest local group, then the launched-to-useful work inflation is*

$$\rho_q = \frac{E_{\mathrm{loc}} \max_j m_{q,j}}{\sum_{j=1}^{E_{\mathrm{loc}}} m_{q,j}}. \tag{31}$$

The equal-load optimum is $\rho_q = 1$, and any imbalance or padding raises the ratio above one.

Proposition B.6 (Routing-context memory) *A compact lower-bound routing-context budget is*

$$M_{\mathrm{ctx}} \approx 8 L_{\mathrm{moe}} \mathrm{DP}TK \tag{32}$$

bytes, before split tables and masks. At $L_{\mathrm{moe}} = 27$, $\mathrm{DP} = 8$, $T = 4096$, $K = 6$, this is roughly 42 MB per GPU. At $L_{\mathrm{moe}} = 60$, $W = 72$, $T = 32{,}768$, $K = 8$, it is roughly 9.1 GB per GPU.

Proposition B.7 (Barrier budget) *If one peer-visible phase barrier performs an $\Theta(W)$ publish/acquire sweep with 50–100 ns per peer touch, then a single barrier costs 3.6–7.2 μs at $W = 72$ and the four barriers in one segment cost roughly 14–29 μs before launch overhead. The median barrier cost at width is small relative to segment transport time.*

Proposition B.8 (Replicated dense lower bound) *For vocabulary size V, hidden size H, L layers, and L_{dense} dense FFN layers of width D_{dense}, a lower bound on replicated dense parameters is*

$$P_{\text{dense}}^{\text{lb}} = VH + 4LH^2 + 3L_{\text{dense}}HD_{\text{dense}}. \tag{33}$$

This omits norms, routers, shared experts, and any untied output head, so it should be interpreted as a lower bound rather than a full fit model.

C Additional Measurements

Single-GEMM upper bound. Single-GEMM MFU on B200 BF16 rises from 7.2% at $\mu = 384$ to 41.5% at $\mu = 3{,}072$ and plateaus around 55–57% for $\mu \geq 24{,}000$, consistent with approaching the HBM roof. These are upper-bound context numbers; the grouped owner receipts in the main text are the mechanism evidence.

Attention scaling. For 27-layer SDPA attention on B200, latency rises from 44 ms at 4K tokens to 330 ms at 32K tokens. A single replica at 256K tokens per step would therefore spend over 20 seconds in attention alone, which explains why dense replication must be paired with many replicas rather than with DP $= 1$.

Per-component forward breakdown. One MoE layer on 8×B200 at the Moonlight shape spends 0.33 ms in pack, 3.47 ms in activation dispatch, 0.08 ms in sideband dispatch, 0.10 ms in unpack, and 2.05 ms in owner grouped GEMM, for 6.03 ms total. Transport therefore accounts for 66.1% of the forward path at that point.

IPC microbenchmarks. At $T = 4096$, $H = 2048$, $E_{\text{loc}} = 8$, and $K = 6$ on 8-GPU IPC, forward p50/p99 are 1.22/1.53 ms and forward+backward p50/p99 are 3.99/4.24 ms.

Fabric microbenchmarks. On the 2-tray / 8-GPU GB300 fabric slice, the corresponding forward p50/p99 are 1.26/1.43 ms and forward+backward p50/p99 are 2.38/2.41 ms.

D Reproducibility

D.1 Benchmark families

All controlled-routing measurements used an internal reference benchmark with 100 warmup iterations and 500 measured iterations in BF16. The controlled ceiling and skew studies used fixed Moonlight expert weights, fixed $E = 64$, $T = 4096$ per rank, and synthetic route tensors injected at the public RDEP operator surface rather than by patching the production router. This is important because the controlled mechanism results are intended to validate the design rather than to modify the production training path.

The overlap-region RDEP run used Moonlight-16B on 8×B200 at 131,072 tokens per step, and the frontier run used the same stack at 262,144 tokens per step. Realized-training receipts are reported

over logged steps 10–200. The public baseline used the baked nightly PyTorch runtime with the Torchtitan DeepSeek-V3 16B recipe at TP $= 2$, EP $= 4$, and `DP_shard` $= 4$ on the same 8-GPU B200 node.

D.2 Fabric validation surfaces

GB300 fabric validation used the same logical RDEP protocol on an internal GB300 cluster in both the 2-tray / 8-GPU slice and the full 18-tray / 72-GPU NVL72 lane. The fabric slice receipts include smoke, dispatch invariants, BF16 gradient parity, and communication microbenchmarks at $T \in \{1024, 4096\}$. The full-width lane includes smoke, dispatch invariants, communication microbenchmarks, and one routed `moe_bf16` sanity step with finite outputs and gradients at every rank.

D.3 Artifact contract

Every main-text quantitative claim in the paper maps to an artifact family consisting of:

1. the exact workload and hardware description;
2. raw timing arrays or per-step logs over the stated window;
3. the summary statistic quoted in the corresponding table;
4. enough metadata to recover the normalization convention, especially for useful TFLOPS and useful MFU.

For distributed latency tables, the quoted statistic is always a max-rank statistic. For grouped receipts, useful work excludes padded launch rows. For realized training, the step window is stated explicitly.

D.4 Publication recipes

The publication surface is intentionally split into public and private recipe families. The Torchtitan baseline is public, so its launch recipe is reproduced verbatim below. The RDEP measurements were collected from a proprietary internal monorepo and internal cluster runbooks, so the paper exposes those runs through an explicit experiment specification rather than through private launch scripts. That specification is sufficient to understand and audit the evidence: each experiment family states the hardware, model shape, routing mode, transport mode, warmup window, measured window, and success criteria.

Controlled-routing grouped mechanism surface. The direct grouped-GEMM receipts in the main text come from a controlled-routing benchmark on one 8×B200 node. The canonical configuration is: Moonlight expert weights; BF16; $T = 4096$ per rank; $H = 2048$; $D_{\text{ff}} = 1408$; $E = 64$; $K = 6$; zero-drop capacity; 100 warmup iterations; 500 measured iterations; and route families `uniform_distinct` and `zipf_distinct` with $\alpha \in \{0.5, 1.0, 1.5, 2.0\}$. Useful grouped-GEMM MFU is normalized by 2.5 PFLOPS/GPU and excludes padded launch rows. The controlled ceiling uses DP $\in \{1, 2, 4, 8\}$ with $E_{\text{loc}} = E/\text{DP}$.

RDEP overlap-region and frontier surfaces. The overlap-region RDEP run uses Moonlight-16B on one 8×B200 IPC node at 131,072 tokens per step with BF16, sequence length 4096, and a 30-step

Table 21 Recipe summary for the main quantitative claims.

Evidence family	Hardware / runtime	Publication surface
Controlled routing	8×B200 IPC, internal RDEP stack	operator shape, route family, warmup and measurement windows, and receipt statistics
Overlap-region RDEP	8×B200 IPC, Moonlight	model shape, token budget, step window, and receipt statistics
Overlap-region baseline	8×B200 IPC, Torchtitan	exact public command line and measurement window
Frontier baseline	8×B200 IPC, Torchtitan	same public command family at the first failing token budget
GB300 fabric slice	2 trays / 8 GPUs	smoke, invariants, BF16 grad parity, transport bench, and routed-compute sanity specification
GB300 NVL72	18 trays / 72 GPUs	smoke, invariants, transport bench, and routed-compute sanity specification

window from which steps 10–29 are reported. The frontier run uses the same configuration family at 262,144 tokens per step. The realized-training receipts use the same Moonlight stack on one 8×B200 node and report logged steps 10–200. These publication surfaces are sufficient to locate the exact rows used in the overlap, frontier, and realized-training tables without reproducing private internal launch wrappers.

GB300 fabric validation surfaces. The initial target-hardware validation used a 2-tray / 8-GPU GB300 slice in fabric mode with $W = 8$ and local tray width 4. The publication surface consists of four receipt families: fabric smoke; dispatch invariants at $T = 256$, $H = 256$, $E_{\text{local}} = 8$, and $K \in \{2, 4\}$; BF16 gradient parity at $T = 16$, $H = 64$, $D_{\text{ff}} = 128$, $E_{\text{local}} = 8$, and $K \in \{2, 4\}$; and communication microbenchmarks at $T \in \{1024, 4096\}$, $H = 2048$, $E_{\text{local}} = 8$, $K = 6$, with 10 warmup and 50 measured iterations for both forward and forward+backward.

The full-width NVL72 validation uses the same logical protocol on 18 trays / 72 GPUs. The publication surface consists of fabric smoke; dispatch invariants at $T = 128$, $H = 128$, $E_{\text{local}} = 1$, and $K \in \{2, 4\}$; communication microbenchmarks at $T \in \{1024, 4096\}$, $H = 2048$, $E_{\text{local}} = 1$, and $K = 6$ with 10 warmup and 50 measured iterations; and one routed `moe_bf16` sanity step at $T = 1024$, $H = 2048$, $D_{\text{ff}} = 1408$, $E_{\text{local}} = 1$, and $K = 6$ with finite outputs and gradients at every rank.

Torchtitan overlap-region and frontier commands. The TP + NCCL baseline used Torchtitan's DeepSeek-V3 16B recipe with the runtime's baked nightly Torch preserved:

Listing 2 Torchtitan TP+NCCL overlap-region and frontier recipes on one 8xB200 node

```
cd /path/to/torchtitan
export PYTHONPATH=$PWD
export PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True

python -m pip install datasets tensorboard tabulate wandb fsspec tyro tokenizers safetensors
python -m pip install --no-deps torchdata

# Overlap-region point: 131,072 tokens/step (global batch 32).
```

```
NGPU=8 LOG_RANK=0,7 \
CONFIG_FILE=./torchtitan/models/deepseek_v3/train_configs/deepseek_v3_16b.toml \
./run_train.sh \
  --model.name deepseek_v3 \
  --model.hf-assets-path ./tests/assets/tokenizer \
  --training.local-batch-size 8 \
  --training.steps 10 \
  --training.dataset c4_test \
  --metrics.log-freq 1 \
  --compile.no-enable \
  --parallelism.pipeline-parallel-degree 1 \
  --parallelism.data-parallel-shard-degree 4 \
  --parallelism.data-parallel-replicate-degree 1 \
  --parallelism.tensor-parallel-degree 2 \
  --parallelism.expert-parallel-degree 4 \
  --parallelism.expert-tensor-parallel-degree 1 \
  --activation-checkpoint.mode none

# Frontier probe: 196,608 tokens/step (global batch 48), expected to OOM.
NGPU=8 LOG_RANK=0,7 \
CONFIG_FILE=./torchtitan/models/deepseek_v3/train_configs/deepseek_v3_16b.toml \
./run_train.sh \
  --model.name deepseek_v3 \
  --model.hf-assets-path ./tests/assets/tokenizer \
  --training.local-batch-size 12 \
  --training.steps 3 \
  --training.dataset c4_test \
  --metrics.log-freq 1 \
  --compile.no-enable \
  --parallelism.pipeline-parallel-degree 1 \
  --parallelism.data-parallel-shard-degree 4 \
  --parallelism.data-parallel-replicate-degree 1 \
  --parallelism.tensor-parallel-degree 2 \
  --parallelism.expert-parallel-degree 4 \
  --parallelism.expert-tensor-parallel-degree 1 \
  --activation-checkpoint.mode none
```