HiPipe: Coordinating Shared and Routed Execution in Mixture-of-Experts Models

Anonymous authors
Paper under double-blind review

Abstract

Shared experts in mixture-of-experts (MoE) models process every token to capture common knowledge, complementing the specialization of routed experts. The shared and routed computations can run concurrently, but the layer cannot finish until both outputs are available. In the distributed schedule we study, the shared expert's first linear projection waits for all input shards, although each shard can be processed independently. We present HiPipe, a hierarchical pipeline that overlaps input transfer with shared projection during routed execution. It projects the local shard first, then overlaps GPU copy-engine transfers of peer shards with projections of ready groups. Configurable grouping balances earlier starts with matrix-multiplication efficiency. Routed work and concurrent execution costs can limit the value of local savings, so we assess groupings by joint completion. We evaluate workloads derived from eleven MoE architectures on H200 and A800. In the repeated A800 study, all eleven shapes improve input projection at 16K and 32K tokens with fixed one-shard groups. At 32K, geometric-mean speedups are $1.416\times$ for input projection and $1.036\times$ for synthetic joint replay over their respective NCCL references. The measured best grouping differs between input projection and joint replay in 27 of 55 workloads. Native-layer and MoE-stack training measurements do not establish corresponding end-to-end gains. These results identify a profitable long-input regime and support choosing grouping and pipeline use by joint completion.

1 Introduction

Mixture-of-experts (MoE) models scale capacity by activating a subset of experts for each token (Shazeer et al., 2017). Shared experts complement this selective computation with a path that processes every token, aiming to capture common knowledge alongside routed specialization (Dai et al., 2024; Yang et al., 2024). The shared and routed paths can run concurrently, but the layer needs both outputs. Any shared work left when routed execution finishes is therefore exposed at layer completion. The native baseline in our experiments already exploits branch-level concurrency; however, this does not remove waiting within each branch. With sequence-sharded inputs, each GPU initially holds only some token rows (Korthikanti et al., 2023). In the schedule we study, the shared expert gathers the remaining rows before its first linear projection. Locally available and early-arriving inputs therefore wait for the entire gather to finish.

This wait is imposed by the schedule: the projection processes token rows independently. Full input availability is required to finish the projection, but not to begin it. Ready shards can thus be projected while the others are transferred, advancing shared work during routed execution (Figure 1). Processing each shard as soon as it arrives reduces waiting but produces smaller matrix multiplications, which may run less efficiently. At short input lengths, reduced projection efficiency and scheduling overhead can outweigh the waiting saved by pipelining. Larger inputs provide more rows per group, making input scale part of the trade-off. Even a faster shared path may not shorten the layer: savings may be masked by longer routed work, and a different shared schedule may change concurrent execution costs. A useful grouping must therefore improve the time when both branches finish.

We present HiPipe, a hierarchical pipeline that retains shared–routed concurrency and applies communication–computation overlap (Chang et al., 2024; Zhang et al., 2025) inside the shared path. HiPipe first projects the local shard, then overlaps GPU copy-engine transfers of peer shards

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

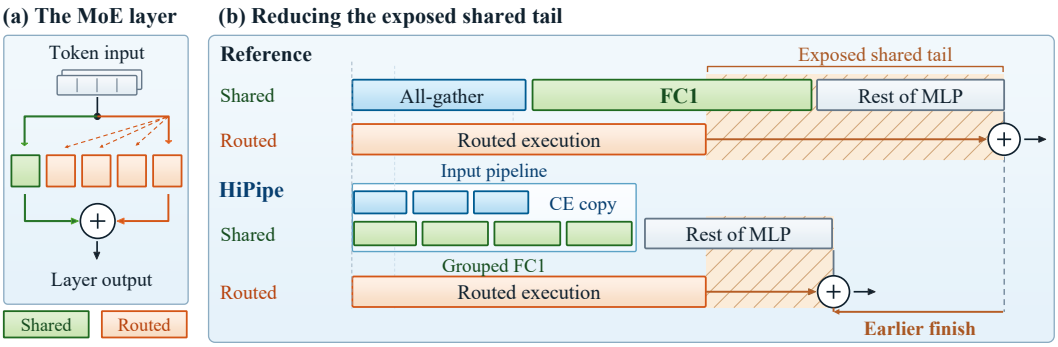


Figure 1: **HiPipe reduces shared work exposed at layer completion.** (a) Each token needs both shared and routed outputs. (b) Within concurrent branch execution, HiPipe overlaps input copies with the first shared projection (FC1). Hatching marks shared work remaining after routed completion. The remaining MLP includes activation, FC2, and reduce-scatter. Timelines illustrate a favorable case with a residual tail; durations are schematic.

with projections of ready groups. Grouping controls both when a projection can start and how many token rows it processes. The projected rows retain their original order for the remaining shared computation. We evaluate this schedule by joint completion, separating exposed local savings from changes in concurrent execution cost. This links the choice of group size to whether the pipeline should be used for a given workload.

Experiments span eleven architecture-derived workloads on H200 and A800, from input projections and concurrent replays to native Megatron MoE layers and four-MoE-layer training stacks. The repeated A800 study identifies a favorable long-input regime: all eleven shapes improve input projection at 16K and 32K with fixed one-shard groups. At 32K, the geometric-mean speedups are $1.416\times$ for input projection and $1.036\times$ for synthetic joint replay, with all eleven shapes improving in both scopes. The measured best grouping changes between these objectives in 27 of 55 workloads. Held-out evaluation further supports choosing by joint latency while retaining conventional execution as an option. Separate native-layer and training measurements delimit the current integration scope; they do not establish corresponding end-to-end gains. These results connect the observed long-input benefit to the decision that matters for the layer: which grouping, and whether to pipeline, advances joint completion.

Our contributions are as follows:

- We develop HiPipe, a hierarchical input pipeline that overlaps copy-engine transfers with projections of ready groups and makes granularity configurable within shared–routed concurrency.
- We characterize how input readiness and projection granularity affect shared-expert execution, separating exposed local savings from changes in concurrent execution cost.
- We evaluate eleven architecture-derived workloads on two GPU platforms, identify a long-input benefit region and different local and joint grouping preferences, and assess held-out choices alongside native-layer and training measurements.

2 Related Work

Shared experts and architectural overlap. Sparsely gated MoE expands capacity through selective expert activation (Shazeer et al., 2017). GShard, Switch Transformers, and GLaM scale this principle to large Transformer models (Lepikhin et al., 2021; Fedus et al., 2022; Du et al., 2022). Residual MoE combines a dense branch with a routed expert (Rajbhandari et al., 2022). DeepSeekMoE separates shared experts for common knowledge from routed specialists (Dai et al., 2024), a design also used by Qwen2's MoE model (Yang et al., 2024). Shared inputs are independent of routing decisions, but both branch outputs are required before the layer can finish. ScMoE and FarSkip-Collective change

cross-layer dependencies to create additional overlap (Cai et al., 2025; Dukler et al., 2026). HiPipe takes the shared-expert architecture as given and schedules its common computation.

Scheduling distributed expert execution. MoE runtimes manage both uneven workloads and communication dependencies. FastMoE supports distributed expert execution on GPUs (He et al., 2021); MegaBlocks uses block-sparse kernels for irregular token assignments (Gale et al., 2023), and FlexMoE adapts expert placement to changing loads (Nie et al., 2023). FasterMoE and Tutel pipeline token exchange with expert computation and adapt execution to workload conditions (He et al., 2022; Hwang et al., 2023). Lancet extends overlap across the training graph through operator partitioning and weight-gradient scheduling (Jiang et al., 2024); MegaScale-MoE combines inter-operator and intra-operator overlap (Jin et al., 2025). FinDEP partitions and orders tasks across separate attention and expert GPU groups, including shared-expert computation (Pan et al., 2025). These systems establish the surrounding schedule within which shared-input progress must be evaluated.

Releasing work as data becomes ready. Tensor parallelism distributes projection weights across GPUs (Shoeybi et al., 2019); sequence parallelism shards token rows and gathers them before a column-parallel projection (Korthikanti et al., 2023). CoCoNeT represents computation and communication together to enable fusion and overlap (Jangda et al., 2022). Domino exposes overlap through tensor slicing (Wang et al., 2024), while FLUX uses tile-readiness checks within GEMM pipelines (Chang et al., 2024) and TileLink generates overlapped kernels from tile primitives (Zheng et al., 2025). For routed MoE, COMET reconciles token movement with computation tiles (Zhang et al., 2025); output-side signaling starts communication as expert results become ready (Cui et al., 2026). In the shared branch, grouping connects input readiness to projection efficiency. HiPipe applies this principle at shard-group granularity, using ordinary matrix multiplications and copy-engine transfers. Its adjustable unit is the number of ready input shards processed as a group.

Resource costs and joint completion. Earlier starts are useful only if concurrent execution advances completion. Lina prioritizes MoE all-to-all under competing communications (Li et al., 2023); Centauri combines communication partitioning with hierarchical scheduling (Chen et al., 2024). TokenWeave examines small-workload partitioning and the GPU resources consumed by communication (Gond et al., 2026). Entwine coordinates tile production and resource allocation for GEMM–ReduceScatter (Ma et al., 2026), while analytical MoE resource management uses GPU residency constraints to allocate communication work (Liu et al., 2026). HiPipe relates exposed shared-path savings and changes in concurrent execution cost to when both branches finish. This makes joint latency the criterion for choosing shared-input granularity. In our replay comparisons, the routed workload remains fixed while the shared schedule changes. This separates an earlier shared completion from a change in the time at which the two paths can be combined.

3 HiPipe

HiPipe replaces the shared expert's gather-then-project schedule with a pipeline that projects groups of inputs as they become available. Figure 2 shows the input transfers and projections running alongside routed-expert computation.

3.1 Shared-expert computation

A shared-expert MoE layer adds the outputs of its shared and routed branches. Both can execute concurrently, but the layer must wait for both outputs. We focus on the input communication before the shared expert's first linear projection (FC1).

Let $X \in \mathbb{R}^{S\times H}$ contain all S token rows of hidden width H. The rows are divided equally across p GPUs, so GPU i initially holds an input shard X_i with S/p rows. The projection weights are partitioned along their output dimension. For the SwiGLU shared expert used here, the first projection produces both gate and up activations, giving a total width of $2D_s$, where D_s is the shared intermediate width. GPU i therefore holds $W_{1,i} \in \mathbb{R}^{H\times(2D_s/p)}$ and computes $Z_i \in \mathbb{R}^{S\times(2D_s/p)}$:

$$Z_i = XW_{1,i}. \tag{1}$$

In the conventional schedule, an all-gather assembles X before this matrix multiplication begins. The weights are already available on GPU i; it is waiting for the missing input rows.

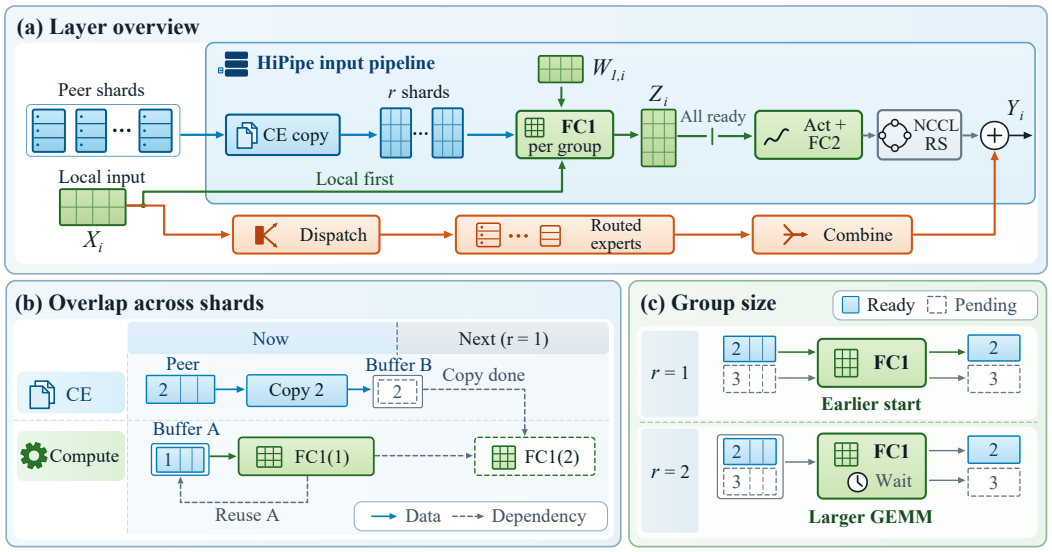


Figure 2: **Input transfer and shared projection in HiPipe.** (a) Local input enters FC1 first; peer copies overlap with grouped FC1 and routed computation. Activation, FC2, and reduce-scatter follow all FC1 groups. Y_i denotes the layer output stored on GPU i. (b) A copy engine fills buffer B while FC1 reads buffer A. Dashed arrows indicate when a buffer can be read or overwritten. (c) Smaller groups start earlier; larger groups supply more rows per matrix multiplication. Two buffers and $r \in \{1, 2\}$ are illustrative.

After the projection, the shared branch applies SwiGLU, a second linear projection (FC2), and NCCL reduce-scatter. The reduction sums contributions from the weight partitions and returns each GPU's token rows. HiPipe retains these operations and any native shared-expert gate; Appendix I.1 gives the full computation.

3.2 Overlapping input transfer and projection

The first projection acts independently on each token row. Divide X into K contiguous row blocks $X^{(1)}, \ldots, X^{(K)}$, listed in their original token order. The superscript indexes row blocks; the subscript i identifies the GPU that owns the weights. Then

$$XW_{1,i} = \begin{bmatrix} X^{(1)}W_{1,i} \\ \vdots \\ X^{(K)}W_{1,i} \end{bmatrix}. \tag{2}$$

Each block needs only its own input rows and the local weight partition. Full input availability is required to finish the projection, but not to begin it.

HiPipe launches the projection of the local shard without waiting for remote inputs. GPU copy engines transfer remote shards into temporary input buffers. Once the inputs for a group have arrived, its projection runs while another buffer receives the next group. Each result is written to its assigned rows in Z_i, preserving token order even when inputs are processed in a different order.

A projection waits for its input copies to finish. A buffer can be filled again only after the preceding projection has finished reading it. CUDA events enforce these two dependencies. Once all groups have completed the first projection, the current implementation proceeds to SwiGLU, the second projection, and reduce-scatter. Appendix A specifies how input buffers remain valid across calls, and Appendix I.1 describes the transfer order and buffer layout.

3.3 Choosing the group size

After launching the local projection, HiPipe groups up to r remote shards for each input group, giving up to rS/p rows. With $r = 1$, a remote shard can be projected as soon as its copy finishes. With $r = 2$, a full group waits for two shards but can process more rows in one matrix multiplication. The last group may contain fewer shards. Groups whose output rows are not contiguous use smaller projection calls (Appendix I.1).

Larger groups delay the start of computation but can improve matrix-multiplication efficiency and reduce the number of launches. Input length also changes this trade-off: it determines how many rows each shard contains. The same group size can therefore behave differently for short and long inputs.

We compare group sizes by the time until both branches finish. Speeding up the shared branch may not shorten this time if routed work finishes last. Concurrent execution can also change branch runtimes, so isolated timings alone cannot predict joint latency. Appendix I.4 relates these measurements.

Our experiments compare fixed group sizes, offline sweeps, and the conventional gather-then-project path to test the choice of timing objective. These are offline comparisons; the calibration rule in Appendix H.2 remains a proposal.

3.4 Integration with Megatron

The current Megatron wrapper copies each invocation's local input into a buffer that peer GPUs can access. It synchronizes before replacing the previous input and after writing the new input, so peer reads use the current invocation's data. The projection kernel also uses a transposed copy of the native weight matrix. The wrapper refreshes this copy on every forward call so that optimizer updates are reflected in the next projection. These steps add input-copy, weight-copy, and synchronization work to each invocation.

An autograd wrapper saves the input and connects the projection to Megatron's original parameters. Backward computes the input gradient with a matrix multiplication followed by reduce-scatter. It reconstructs the complete input with an NCCL all-gather to compute the first projection's weight gradient. The resulting gradients are returned to the original parameter tensors. The backward equations, numerical checks, and training configurations are reported in Appendices F, J.2, and J.4, respectively.

4 Experiments

We evaluate where earlier shared projection reduces latency and which scheduling choices make that progress useful. The experiments first identify a benefit region, then examine the transfer schedule and group size, and finally test whether local improvements carry over to joint and native execution.

Workloads and measurement. Our main study uses shapes derived from eleven MoE architectures on eight A800 GPUs. It varies input length from 2K to 32K tokens and repeats each workload in five launches. We measure input projection and synthetic joint replay separately; the latter runs the shared path alongside a dense proxy for routed work. Backend controls, native Megatron MoE layers, and four-MoE-layer training stacks provide complementary tests. Historical H200 and A800 campaigns extend the replay coverage but use different implementations and protocols, so we report them separately. Appendices B and J give the workload definitions and measurement scopes.

Comparisons. The input reference is NCCL all-gather followed by a full projection; branch and layer references already overlap shared and routed work. Unless specified otherwise, HiPipe uses $r = 1$, three buffers, and one copy and one compute stream. Repeated A800 forward runs use 50 warm-up and 200 timed samples. We take the maximum across ranks per sample, the p50 within each launch, and the median across launches for latency. Speedups are geometric means of paired within-launch ratios, using each scope's own reference. Reported ranges span launches, not confidence intervals. Pairing within launches ties each speed comparison to a common launch before aggregation. Appendix A states the execution requirements and the validation limits of the historical records.

Table 1: **Long-input gains across all eleven A800 workload shapes.** TP8, fixed $r = 1$, full gate-and-up width. Values geometrically average five within-launch NCCL/HiPipe ratios using each scope's own reference. Joint replay uses a dense routed-work proxy. Counts use unrounded point estimates; numerical-check coverage is documented in Appendix J.3.

Derived workload	Input projection		Joint replay	
	16K	32K	16K	32K
Qwen2	1.216	1.228	1.019	1.017
Hunyuan	1.193	1.168	1.024	1.017
Llama4	1.109	1.407	1.000	1.030
Qwen1.5	1.405	1.549	1.021	1.057
DS-V4-Pro	1.548	1.613	1.046	1.048
MiniMax-M3	1.523	1.610	1.045	1.045
DS-V2-Lite	1.240	1.412	0.989	1.041
DS-V3	1.411	1.426	1.024	1.037
DS-V4-Flash	1.323	1.393	1.018	1.030
GLM-5.2	1.407	1.421	1.030	1.036
Hy4-preview	1.401	1.420	1.030	1.036
All shapes (GM)	**1.337**	**1.416**	**1.022**	**1.036**
Shapes above $1\times$	11/11	11/11	9/11	11/11

4.1 Long inputs provide a consistent benefit region

Long inputs give a common benefit region in the recorded A800 timings, while short inputs do not reliably repay pipelining. At both 16K and 32K, every shape improves its input projection with fixed $r = 1$ (Table 1). At 32K, all eleven also improve synthetic joint execution: the geometric-mean speedups are $1.416\times$ for input projection and $1.036\times$ for joint replay. These are gains over each scope's conventional reference, with no per-shape choice of group size. The agreement across shapes makes the long-input result broader than a gain obtained by tuning one favorable workload.

The improvement develops with input length rather than appearing uniformly across the sweep. At 2K, the mean input-stage ratio is $0.850\times$ and most shapes are slower than the reference. At 8K, input projection improves on average, while joint replay remains slightly slower (Figure 4). By 32K, the gain extends to both scopes for every shape. Thus the long-input result establishes where the pipeline is useful in this study, while the shorter inputs show why a local speedup alone is insufficient to select a joint schedule. The full sweep, including the transition at 4K, is retained in Appendix J.3. Input length thus helps identify candidates for pipelining, but the different crossovers in the two scopes require checking each candidate at joint completion. The observed region does not define a universal token threshold.

Historical layer replays provide a broader view of workload dependence (Figure 3). Favorable cases occur on both GPUs, alongside regressions at other shapes and lengths. Because these campaigns differ in implementation and environment, the panels describe coverage within each campaign rather than isolate a hardware effect. Full latencies and the additional routed-communication replays appear in Appendices K and G.1. Together, the sweeps show that a common long-input benefit can coexist with regressions in other execution settings. Input length identifies where to look for a gain; the measured schedule determines whether to use the pipeline.

4.2 The transfer schedule and group size both matter

The controls support pipelining ready-input projection rather than replacing the transfer backend alone. They also show that larger groups help only some workloads. We next examine which scheduling choices matter within a fixed comparison. Increasing input length adds token rows without changing a workload's projection width, so Figure 4 varies the work per shard while keeping the weights fixed. The following comparisons separate the transfer choice from the decision about when and how much input to project.

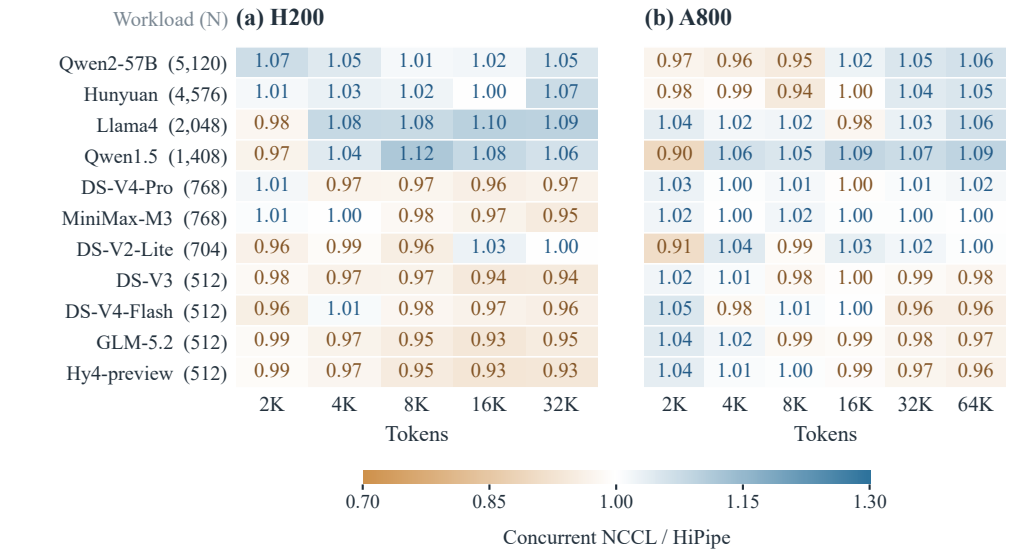


Figure 3: **Layer-replay gains vary with workload and input length.** Concurrent-NCCL/HiPipe latency ratios for (a) 55 H200 cases at 2K–32K and (b) 66 A800 cases at 2K–64K; values above one favor HiPipe. Rows follow projection width $N = 2D_s/8$. The campaigns differ in environment and implementation. Figure 5 adds A800's 512 and 1K cases.

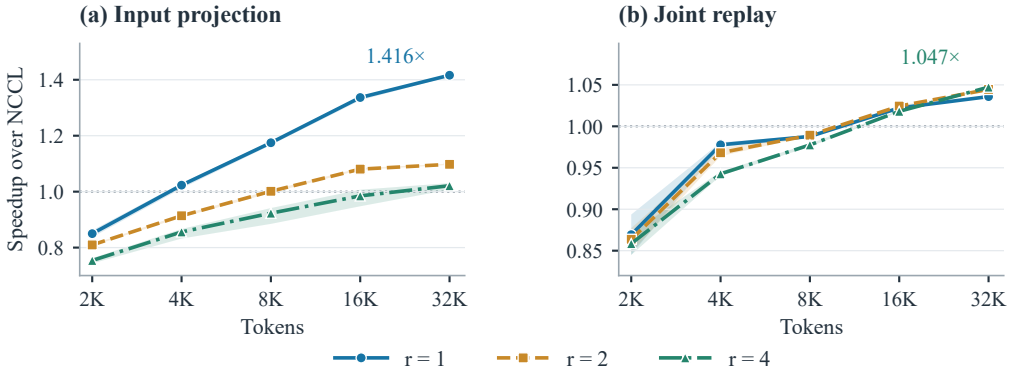


Figure 4: **The best grouping depends on where completion is measured.** A800, TP8: (a) input projection and (b) synthetic joint replay. Lines show geometric-mean NCCL/HiPipe ratios across eleven shapes and five launches; shading spans the five launch-level shape means, not confidence intervals. The panels use different vertical scales. At 32K, $r = 1$ maximizes the input-stage mean, whereas $r = 4$ maximizes the joint-replay mean.

Changing the transfer backend is insufficient. In the separate backend-control batch, CE gather-then-project is slower than NCCL in all six workloads (Table 2). For Qwen1.5 at 32K, NCCL takes 1.613 ms and CE gather takes 3.394 ms, whereas HiPipe takes 1.043 ms. Using copy engines alone therefore does not account for the benefit in this comparison: the gain appears when input transfer is combined with early projection. The implementations also differ in transfer organization, so this control does not isolate the amount of communication hidden by overlap. The supported conclusion is that transfer and projection must be evaluated as a schedule; copy-engine use alone is insufficient to explain these gains.

Larger projections must repay the later start. Combining two remote shards helps Qwen2 at 8K but hurts DeepSeek-V3 at 32K (Table 2). More rows per call can improve matrix-multiplication

Table 2: **A800 input-projection controls.** Median launch-level p50 latency (ms), five launches, TP8, and gate-and-up width $N = 2D_s/p$. NCCL and CE gather complete transfer before one projection; HiPipe projects groups of r shards. CE timings include input publication. Appendix J.3 gives paired ratios and complete coverage.

Shape	Tokens	NCCL	CE gather	HiPipe		
				$r = 1$	$r = 2$	$r = 4$
Qwen1.5	8K	0.473	0.916	0.420	0.495	0.552
Qwen1.5	32K	1.613	3.394	1.043	1.370	1.475
Qwen2	8K	1.547	2.307	1.586	1.413	1.605
Qwen2	32K	6.013	9.003	4.903	4.901	5.222
DeepSeek-V3	8K	1.152	2.716	0.856	1.111	1.143
DeepSeek-V3	32K	3.860	10.292	2.722	3.951	4.080

efficiency, but the group also waits for more input. The measured outcome depends on this balance, not simply on making each projection larger. Single-shard groups give the best input-stage point estimate in 45 of the 55 repeated A800 workloads, while the remaining workloads favor larger groups. This result favors small groups as a useful starting point for this sweep, while leaving group size as a measured choice.

Additional resources likewise do not guarantee a better schedule. In the H200 configuration sweep, a second compute stream or a fourth staging buffer changes aggregate latency little (Table 7). The separate TE comparison also shows why a conventional reference remains necessary: outperforming another overlap implementation need not outperform gather-then-project. That comparison uses a narrower projection and is reported separately in Appendix E. Together, these controls support choosing granularity from measured execution rather than from group size or resource count alone. The observed gains depend on how transfers and projections are scheduled together, rather than on maximizing the resources assigned to either operation.

4.3 Group size must be evaluated at joint completion

The group that minimizes input-projection time can be suboptimal at joint completion. We test this by comparing the same measured groupings under both objectives. At 32K, $r = 1$ gives the best mean input-stage ratio, whereas $r = 4$ gives the best mean joint-replay ratio (Figure 4). Increasing the group size reduces the input-stage gain but raises the joint mean from $1.036\times$ to $1.047\times$; the joint ordering holds in all five launch-level means. Across the complete sweep, the input and joint point-estimate optima differ in 27 of 55 workloads. This reversal makes joint timing the selection criterion and gives isolated input timing a diagnostic role: it explains shared progress without determining the final choice.

Historical H200 branch records show that concurrency can amplify or reverse shared-stage savings. For Llama4 at 16K, joint execution saves more time than the shared branch measured alone. For DeepSeek-V3 at the same length, the shared branch becomes faster while joint execution becomes slower (Appendix C). In 25 of the 26 recorded reversals, optimized shared work still takes longer than routed work when measured separately. A slower routed branch masking the saving therefore cannot explain most reversals. The residual change under concurrency outweighs a saving that remains exposed in the isolated-time model. Appendix I.4 separates these contributions without assigning a hardware cause. Joint measurements with routed work present are therefore necessary to judge usefulness. The isolated measurements help locate the reversal, while the concurrent measurement determines whether the candidate actually advances completion.

Choosing a group also requires deciding whether to pipeline. Joint-based selection works best in this offline comparison when it can retain conventional execution. We test the choice criterion by selecting a configuration on four A800 launches and evaluating it on the excluded fifth launch. Rotating this split covers every workload and launch (Appendix H.1). Joint-based selection improves on input-based selection, but both remain slower than NCCL on average when a pipeline configuration is mandatory. Allowing the conventional path yields $1.018\times$ for joint-based selection, compared with

$1.003\times$ for input-based selection. The result links the long-input gains and short-input regressions: a useful policy must select both the group size and whether to enable pipelining. This is an offline comparison of recorded times; it excludes calibration costs and does not establish a deployed selector. Native execution must test the choice with the full MoE computation present. Within this offline test, enabling the pipeline and choosing its group size are two parts of the same joint-latency decision.

4.4 Shared-parameter checks and native execution

Matched native tests establish elementwise agreement for the shared parameters. Four single-layer configurations compare serial, overlap, and HiPipe execution from the same initial weights and inputs. The comparison covers all eight ranks, four forward/backward passes, and separate five-step Adam checks. Shared-parameter gradients, updates, and updated weights agree elementwise. Elementwise errors are computed before their worst values are taken across steps and ranks. Averaging gradient norms could conceal differences between parameters or devices, whereas these comparisons retain the largest recorded discrepancy. The optimizer check compares both the update and the resulting parameter value, connecting a backward-pass comparison to the state consumed by the next forward pass. Cosine similarity is reported as a complementary measure: values close to one describe alignment, while the elementwise errors determine the shared-parameter agreement reported here.

Agreement is specific to those tensors and the tested summed-output loss. Rank-local losses are identical in 157 of 160 comparisons; input-gradient relative L_2 differences reach 6.0×10^{-4}, comparable in scale to overlap–serial differences. Some output and input-gradient entries exceed the recorded elementwise tolerances despite cosine similarities close to one (Appendix J.2). Thus shared-parameter agreement and full-layer numerical agreement are different outcomes, and the comparison reports them separately.

Native latency is also a separate measurement objective. These tests execute actual routing and expert computation, whereas joint replay uses a dense routed-work proxy. Complete layer time includes dispatch, both expert paths, and output combination. The repeated A800 batch does not show a consistent layer speedup: Qwen2 approaches parity at long inputs, while Qwen1.5 and DeepSeek remain slower (Table 17). Four supplementary layer timings agree with this pattern (Table 20). Historical H200 training results are mixed, and the repeated A800 four-layer study does not establish training acceleration (Tables 9 and 24). These comparisons show why projection gains must be evaluated at the native layer boundary before drawing a training-level conclusion.

Staging also introduces a measurable storage cost. Historical A800 operator runs add 176.0–624.1 MiB at 32K across the eleven shapes (Appendix G); Table 25 reports training peaks. Native evaluation must account for buffer storage together with input publication, weight maintenance, and backward communication. The complete execution, including these operations, is therefore the unit for judging an integrated schedule; a faster isolated projection alone does not determine its value.

5 Conclusion

HiPipe advances shared-expert computation by projecting available token rows while the remaining inputs are transferred. Grouping makes the resulting trade-off explicit: earlier starts must preserve enough work per projection to use the GPU efficiently. The repeated A800 study identifies a long-input region where this approach helps across all eleven shapes, with gains in both input projection and synthetic joint replay at 32K.

The experiments also show why input-stage efficiency is only part of the scheduling decision. The group that minimizes projection time can differ from the group that minimizes joint time, and shorter inputs can favor conventional execution. These findings connect input readiness, computation size, and the completion of the surrounding MoE schedule. The native-layer and training measurements place these gains in context: a schedule that improves shared projection must also advance the surrounding computation to reduce total latency. The design lesson is to advance ready shared work at a granularity chosen for joint completion, and to retain the conventional path where pipelining does not repay its costs.

AI USE STATEMENT

Generative AI tools assisted with literature review, drafting, translation, figure preparation, scheduling analysis, experiment-design feedback, analysis code, and interpretation of recorded results. References and numerical summaries were checked against source papers and archived measurements; tables were regenerated from those records, and the manuscript and figures were compiled and inspected. The authors take responsibility for the final text, code, and claims, including AI-assisted contributions.

REPRODUCIBILITY STATEMENT

Appendices B and J specify workloads, timing scopes, aggregation, and the recorded environment. The accompanying archive provides raw measurements, per-launch aggregates, table generators, and source manifests. Appendix J.2 gives the matched single-layer gradient and update checks, including outputs, input gradients, per-rank losses, and the exact comparison criteria. Each experimental campaign is reported with its own implementation and measurement scope.

REFERENCES

Weilin Cai, Juyong Jiang, Le Qin, Junwei Cui, Sunghun Kim, and Jiayi Huang. Shortcut-connected expert parallelism for accelerating mixture of experts. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 6211–6228. PMLR, 2025. URL https://proceedings.mlr.press/v267/cai25c.html.

Li-Wen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Chengji Yao, Ziheng Jiang, Haibin Lin, Xin Jin, and Xin Liu. FLUX: Fast software-based communication overlap on GPUs through kernel fusion. *arXiv preprint arXiv:2406.06858*, 2024. URL https://arxiv.org/abs/2406.06858.

Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 178–191, 2024. doi: 10.1145/3620666.3651379. URL https://doi.org/10.1145/3620666.3651379.

Minyu Cui, Anna Wingkvist, and Morgan Ericsson. Fine-grained computation-communication overlap via tile-level signaling and scheduling for mixture-of-experts. *arXiv preprint arXiv:2607.19539*, 2026. URL https://arxiv.org/abs/2607.19539.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.70. URL https://aclanthology.org/2024.acl-long.70/.

Nan Du, Yanping Huang, Andrew M. Dai, et al. GLaM: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5547–5569. PMLR, 2022. URL https://proceedings.mlr.press/v162/du22c.html.

Yonatan Dukler, Guihong Li, Deval Shah, Jiang Liu, Vikram Appia, and Emad Barsoum. FarSkip-Collective: Unhobbling blocking communication in mixture of experts models. In *Proceedings of Machine Learning and Systems*, volume 8, pp. 1803–1816, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/6feb9b30798abcfae937760d183605e1-Abstract-Conference.html.

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. URL https://www.jmlr.org/papers/v23/21-0998.html.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient sparse training with mixture-of-experts. In *Proceedings of Machine Learning and Systems*, volume 5, pp. 288–304, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/5a54f79333768effe7e8927bcccffe40-Abstract-mlsys2023.html.

Raja Gond, Nipun Kwatra, and Ramachandran Ramjee. TokenWeave: Efficient compute-communication overlap for distributed LLM inference. In *Proceedings of Machine Learning and Systems*, volume 8, pp. 36–56, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/73ba81c7b25134a559c8a9c39ec1a4c3-Abstract-Conference.html.

Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. FastMoE: A fast mixture-of-expert training system. *arXiv preprint arXiv:2103.13262*, 2021. URL https://arxiv.org/abs/2103.13262.

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*, pp. 120–134, 2022. doi: 10.1145/3503221.3508418. URL https://doi.org/10.1145/3503221.3508418.

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. In *Proceedings of Machine Learning and Systems*, volume 5, pp. 269–287, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/5616d34cf8ff73942cfd5aa922842556-Abstract-mlsys2023.html.

Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 402–416, 2022. doi: 10.1145/3503222.3507778. URL https://doi.org/10.1145/3503222.3507778.

Chenyu Jiang, Ye Tian, Zhen Jia, Shuai Zheng, Chuan Wu, and Yida Wang. Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 74–86, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/339caf45a6fa281cae8adc6465343464-Abstract-Conference.html.

Chao Jin, Ziheng Jiang, Zhihao Bai, et al. MegaScale-MoE: Large-scale communication-efficient training of mixture-of-experts models in production. *arXiv preprint arXiv:2505.11432*, 2025. URL https://arxiv.org/abs/2505.11432.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. In *Proceedings of Machine Learning and Systems*, volume 5, pp. 341–353, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/80083951326cf5b35e5100260d64ed81-Abstract-mlsys2023.html.

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. URL https://arxiv.org/abs/2006.16668.

Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed MoE training and inference with Lina. In *2023 USENIX Annual Technical Conference*, pp. 945–959.

USENIX Association, 2023. URL https://www.usenix.org/conference/atc23/presentation/li-jiamin.

Hongyu Liu, Minyu Cui, and Miquel Pericas. Analytical resource management for fine-grained MoE computation-communication overlap. *arXiv preprint arXiv:2609.07536*, 2026. URL https://arxiv.org/abs/2609.07536.

Kai Ma, Quanfeng Lv, Jingguo Ge, Bowei Dai, and Kefan Ruan. Entwine: Coordinating tiled computation and fine-grained communication across GPUs. *arXiv preprint arXiv:2609.11562*, 2026. URL https://arxiv.org/abs/2609.11562.

Xiaonan Nie, Xupeng Miao, Zilong Wang, Zichao Yang, Jilong Xue, Lingxiao Ma, Gang Cao, and Bin Cui. FlexMoE: Scaling large-scale sparse pre-trained model training via dynamic device placement. *Proceedings of the ACM on Management of Data*, 1(1):1–19, 2023. doi: 10.1145/3588964. URL https://doi.org/10.1145/3588964.

Xinglin Pan, Shaohuai Shi, Wenxiang Lin, Yuxin Wang, Zhenheng Tang, Wei Wang, and Xiaowen Chu. Efficient MoE inference with fine-grained scheduling of disaggregated expert parallelism. *arXiv preprint arXiv:2512.21487*, 2025. URL https://arxiv.org/abs/2512.21487.

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*, pp. 18332–18346. PMLR, 2022. URL https://proceedings.mlr.press/v162/rajbhandari22a.html.

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. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

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. URL https://arxiv.org/abs/1909.08053.

Guanhua Wang, Chengming Zhang, Zheyu Shen, Ang Li, and Olatunji Ruwase. Domino: Eliminating communication in LLM training via generic tensor slicing and overlapping. *arXiv preprint arXiv:2409.15241*, 2024. URL https://arxiv.org/abs/2409.15241.

An Yang, Baosong Yang, Binyuan Hui, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024. URL https://arxiv.org/abs/2407.10671.

Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, Quan Chen, and Xin Liu. COMET: Fine-grained computation-communication overlapping for mixture-of-experts. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/e27ea0cd50b798ff8942caf9203f0992-Abstract-Conference.html.

Size Zheng, Jin Fang, Xuegui Zheng, Qi Hou, Wenlei Bao, Ningxin Zheng, Ziheng Jiang, Dongyang Wang, Jianxi Ye, Haibin Lin, Li-Wen Chang, and Xin Liu. TileLink: Generating efficient compute-communication overlapping kernels using tile-centric primitives. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/c6ee784cbe46d854843e4c883a3321ef-Abstract-Conference.html.

A Execution Contract

The row decomposition in Equation 2 describes the intended operator. A correct implementation additionally needs the following ownership chain for every invocation: publish the current local input, make that publication visible to peer readers, preserve it until their reads complete, and preserve each assembled output until its downstream consumers finish. Copy-completion and buffer-consumption events establish the local staging dependencies shown in Figure 2. Cross-process publication is a separate dependency.

The historical operator and replay records have a separate provenance issue. Their inspected call sites register an IPC-visible `ag_buf`, while the forward interface receives `x_local`; per-invocation publication into the exported storage and peer readiness are not evident from those call sites. This remains an unresolved contract for those evaluated revisions, not a demonstrated numerical failure.

The newer A800 harness explicitly copies the current input into exported storage and uses retirement and readiness fences. The four Qwen1.5 TP8 configurations in Table 17 have matching forward-check receipts that pass the recorded changing-input and changing-weight tests. Other native configurations retain their individual recorded status in Appendix J.1. These receipts do not retrospectively validate the historical implementations or establish training equivalence. Matched single-layer shared-gradient and update comparisons are reported in Appendix J.2; they do not replace the forward receipts.

The final artifact should pair the measured revision with forward error against the conventional projection over changing inputs, including consecutive invocations that exercise staging reuse. It should include the required publication work in the measured interval. Training extends the same contract to current weights, gradient destinations, optimizer updates, and the lifetime of activations saved for backward.

B Workloads and Measurement Protocol

Table 3: **Dimensions of the eleven architecture-derived workloads.** D_s is the aggregate shared intermediate width, and $N = 2D_s/8$ is the local gate-and-up width at TP8. Names identify recorded workload configurations; they do not imply execution of a model checkpoint.

Workload label	H	D_s	$2D_s/8$
Qwen1.5-MoE	2,048	5,632	1,408
Qwen2-57B	3,584	20,480	5,120
Hunyuan-Large	6,400	18,304	4,576
Llama4-Maverick	5,120	8,192	2,048
DeepSeek-V4-Pro	7,168	3,072	768
MiniMax-M3	6,144	3,072	768
DeepSeek-V2-Lite	2,048	2,816	704
DeepSeek-V3	7,168	2,048	512
DeepSeek-V4-Flash	4,096	2,048	512
GLM-5.2	6,144	2,048	512
Hy4-preview	6,144	2,048	512

Historical operator. The H200 sweep uses BF16 inputs, ten warm-up iterations, and fifty timed iterations with CUDA events in the inspected v4 benchmark. Each record compares sequential NCCL all-gather plus one projection with the CE pipeline. Nine configurations span a specified subset of grouping, buffer, and stream choices, listed in Appendix D. The baseline is timed separately for each configuration. Stored ratios therefore include variation in both reference and candidate measurements.

Historical branch. The shared replay contains AG, first projection, a slice standing in for the activation output, second projection, and RS. Routed work is represented by two matrix multiplications with a token count derived from top-k. It omits a real router, permutation, expert all-to-all, and distinct expert parameter sets. Both branches are measured alone, sequentially, and concurrently. These records characterize that replay's execution; they do not substitute for a complete routed-expert layer.

Table 4: **Historical layer-replay summaries across input ranges.** Each entry geometrically averages concurrent-NCCL/HiPipe ratios over the indicated lengths; values above one favor HiPipe. All eleven shapes and all measured lengths are retained. The campaigns use TP8 and fixed configurations, with their distinct execution-contract qualifications described above. These cross-length aggregates complement the long-input A800 study in Table 1; they measure a different scope.

Derived workload	H200	A800		
	2K–32K	2K–32K	2K–64K	512–64K
Qwen2-57B	1.040	0.990	1.002	0.963
Hunyuan	1.028	0.992	1.002	0.960
Llama4	1.065	1.016	1.023	0.987
Qwen1.5	1.053	1.032	1.042	0.991
DS-V4-Pro	0.976	1.010	1.011	0.993
MiniMax-M3	0.981	1.009	1.008	0.996
DS-V2-Lite	0.987	0.998	0.999	0.927
DS-V3	0.961	0.999	0.995	0.982
DS-V4-Flash	0.974	1.000	0.993	0.941
GLM-5.2	0.958	1.003	0.997	0.985
Hy4-preview	0.956	1.002	0.996	0.973
All shapes (GM)	0.997	1.005	1.006	0.972
Configurations	55	55	66	88

Historical layer. The layer replay prepends attention-shaped projections and collectives. It omits full attention and complete model activations. We use the unified H200 rerun with all five token counts, replacing the earlier standalone 4K run. All results refer to measured per-layer intervals; multiplying them by a nominal depth is not treated as a model or training measurement.

Historical aggregation. The replay scripts save rank-zero summaries. They do not provide a distribution of cross-rank maximum latency over independent runs. For speedup s_d on case d, tables use $\exp(|\mathcal{D}|^{-1} \sum_{d \in \mathcal{D}} \log s_d)$. Recorded speedup fields retain their saved precision. Figure 6 computes both ratios from the paired millisecond fields so its coordinates refer to the displayed times. Values below one are retained throughout.

Complete replay coverage. Figure 5 preserves all available input lengths: 2K–32K on H200 and 512–64K on A800. Main Figure 3 shows H200 at 2K–32K and A800 at 2K–64K, using only measured lengths. The A800 512 and 1K cases remain part of the full-range aggregate.

C The Complete Branch Distribution

Figure 6 places the two illustrative 16K cases discussed in Section 4 within all 55 branch records. Both coordinates refer to the same NCCL-versus-HiPipe comparison, evaluated either on the isolated shared branch or on the joint schedule. There are 26 records with a faster shared path and a slower joint path; no significance threshold is applied to this descriptive count. Table 28 preserves shared-alone and joint milliseconds for every configuration, including Qwen1.5's 2.544× record at 4K and all cases below one. In 25 of the 26 reversals, the optimized shared path remains slower than the routed path in isolation. For the remaining case, the exposed saving is still positive. Thus all 26 reversals require an increase in the measured concurrency residual greater than the exposed saving; they cannot be accounted for solely by eliminating that saving at the isolated-time maximum.

For DeepSeek-V3 at 16K, isolated routed time is 1.224 ms. The conventional concurrency residual is $2.309 - \max(1.595, 1.224) = 0.714$ ms, while HiPipe's is $2.555 - \max(1.441, 1.224) = 1.114$ ms. Its increase of 0.400 ms exceeds the isolated shared saving of 0.154 ms. Equation 9 locates the difference in the change under concurrent execution. Separating its scheduling and resource components requires the matched trace described in Appendix H.

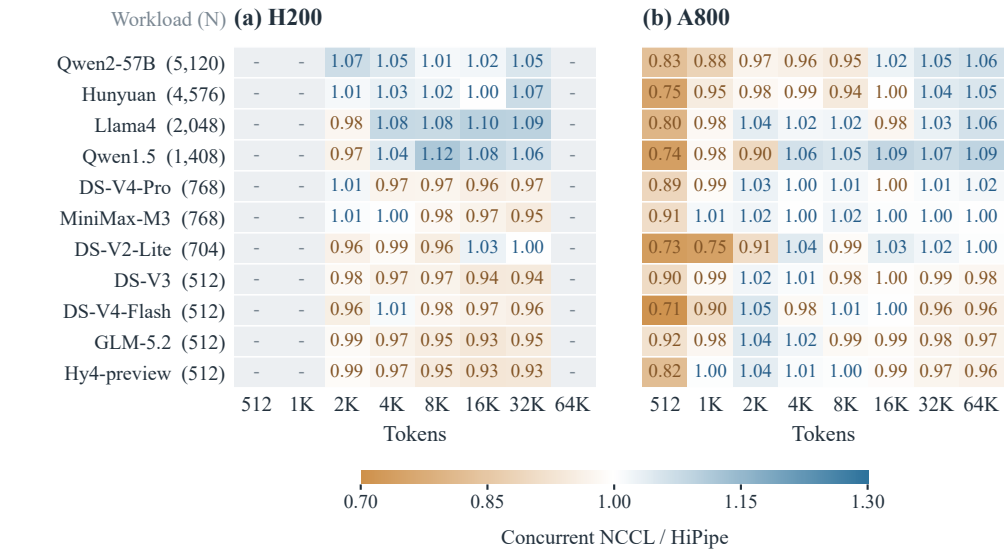


Figure 5: **Complete historical layer-replay coverage.** Concurrent-NCCL/HiPipe latency ratios for all (a) 55 H200 and (b) 88 A800 cases. Values above one favor HiPipe. Rows and the 0.70–1.30 color scale match Figure 3. Gray cells mark unmeasured H200 lengths. The A800 panel includes the additional 512 and 1K workloads.

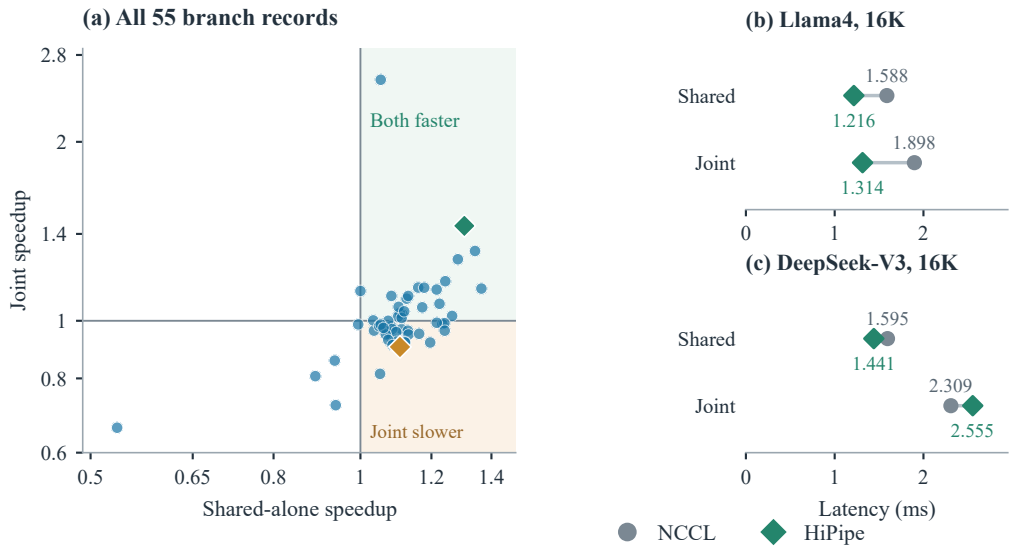


Figure 6: **Shared-alone gains need not reduce joint latency.** (a) All 55 H200 branch records, TP8, with NCCL/HiPipe ratios on logarithmic axes. Diamonds highlight the 16K cases detailed in (b) Llama4 and (c) DeepSeek-V3. The right panels show shared-alone and joint latencies separately; circles denote NCCL and diamonds HiPipe. These point estimates characterize the recorded schedules without isolating a hardware cause.

D Operator Configuration and Complete Scope Summary

The default uses $(r, n_b, n_c, n_{cp}) = (1, 3, 1, 1)$ for shards per group, staging buffers, compute streams, and copy streams. The eight other configurations change, respectively: $r = 2$; $r = 4$; $n_b = 2$;

$n_b = 4$; $n_c = 2$; $n_{cp} = 2$; $(n_c, n_{cp}) = (2, 2)$; and $(r, n_c, n_{cp}) = (2, 2, 2)$. All unspecified values retain the default.

Table 5: **H200 speedups across execution scopes.** Geometric means over five token counts. Operator ratios use sequential all-gather plus projection; branch and layer ratios use concurrent NCCL. †Best selects the largest stored ratio among nine configurations per workload, each with its own timed reference. It is an in-sample summary, not a measured deployment policy.

Workload	N	Op. fixed	Op. best†	Branch	Layer
Qwen1.5-MoE	1,408	0.991	1.137	1.205	1.053
Qwen2-57B	5,120	1.298	1.348	1.028	1.040
Hunyuan-Large	4,576	1.422	1.461	1.038	1.028
Llama4-Maverick	2,048	1.133	1.194	1.250	1.065
DeepSeek-V4-Pro	768	0.847	0.944	0.988	0.976
MiniMax-M3	768	0.860	0.915	0.992	0.981
DeepSeek-V2-Lite	704	0.833	0.975	1.000	0.987
DeepSeek-V3	512	0.777	0.855	0.936	0.961
DeepSeek-V4-Flash	512	0.796	0.863	0.886	0.974
GLM-5.2	512	0.788	0.863	0.925	0.958
Hy4-preview	512	0.787	0.842	0.943	0.956

Table 6: **Input-stage latency under grouping and stream changes.** Recorded milliseconds at 8K tokens on H200, TP8. The sequential column uses the default configuration's reference. Other columns vary shards per group or compute streams; all remaining settings retain their defaults. Absolute times are shown because each source row times its own reference.

Workload at 8K	Sequential	Group 1	Group 2	Group 4	2 comp. streams
Qwen1.5-MoE	0.193	0.183	0.174	0.170	0.172
Qwen2-57B	0.572	0.410	0.412	0.482	0.405
Hunyuan-Large	0.929	0.608	0.621	0.726	0.606
DeepSeek-V3	0.418	0.569	0.519	0.514	0.528

These columns should be read within their stated scope. A small input-stage ratio can coexist with a favorable whole-branch ratio because the measured intervals and concurrent schedules differ. Establishing a causal decomposition requires matched component timings from one execution. The primary design comparison therefore uses joint time directly.

Table 7: **The complete nine-configuration H200 sweep.** Each row aggregates the same 55 workloads. Compute and Copy are stream counts. Latency/default is the geometric mean of candidate/default pipeline times (lower is better); Speedup/NCCL uses each record's reference (higher is better). The last column counts speedup estimates above one.

r	Buffers	Compute	Copy	Latency/default	Speedup/NCCL	> 1
1	3	1	1	1.000	0.936	19/55
2	3	1	1	0.969	0.964	19/55
4	3	1	1	0.976	0.958	21/55
1	2	1	1	1.010	0.924	16/55
1	4	1	1	0.996	0.936	19/55
1	3	2	1	1.000	0.934	17/55
1	3	1	2	0.986	0.948	19/55
1	3	2	2	0.971	0.963	19/55
2	3	2	2	0.959	0.971	20/55

The default's all-workload speedup is $0.936\times$. Selecting the largest observed ratio among nine configurations for each workload raises this descriptive mean to $1.017\times$. That in-sample maximum includes reference-timing variation and selection on the reported measurements; it is not the performance of a

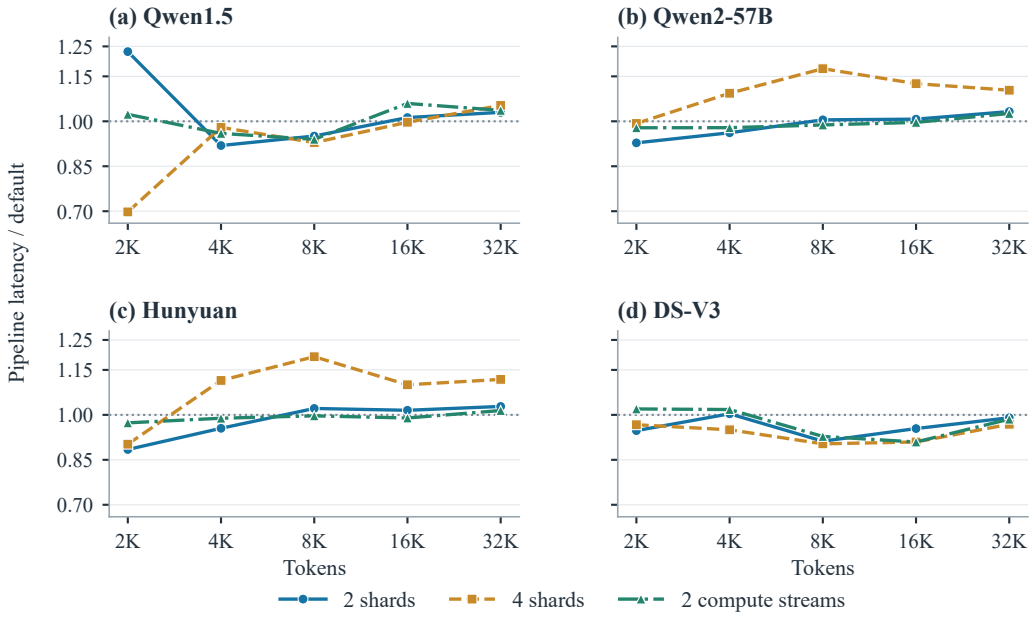


Figure 7: **Grouping and stream effects depend on the H200 workload.** Pipeline latency relative to the default configuration for (a) Qwen1.5, (b) Qwen2-57B, (c) Hunyuan, and (d) DeepSeek-V3. Each curve varies shards per group or compute streams, holding other settings fixed. Values below one favor the variant. Historical point estimates are shown; independent-run intervals are unavailable for this sweep.

deployed selection rule. Directly comparing pipeline milliseconds to the default shows that more streams or buffers alone do not reliably reduce time.

E Official Transformer Engine Comparison

The official-TE benchmark invokes `Linear` with userbuffer all-gather overlap and ring exchange, using `use_ce=False` or `True`. The project record identifies TE 2.14.1 for this H200 run. The script uses four SMs for TE-SM, one for TE-CE, and p splits; these are evaluated settings rather than an exhaustive tuning result. It uses wall-clock timing with synchronization after each iteration.

The TE and primary sweep results describe different linear shapes. The final full-width study will preserve equivalent dimensions, execution and gradient modes, synchronization, and tuning budgets across sequential, TE-CE, NCCL-CE, and HiPipe. The earlier project experiment labeled "TE-equivalent" used NCCL primitives and is not substituted for the official TE results here.

Across all 55 configurations, geometric-mean ratios are $1.581\times$ against TE-SM, $1.325\times$ against TE-CE, and $0.920\times$ against sequential NCCL. The largest TE-SM ratio is Qwen1.5 at 2K: 0.3781 ms for TE-SM versus 0.0969 ms for HiPipe ($3.902\times$), while the sequential reference takes 0.0887 ms. The largest TE-CE ratio is DeepSeek-V2-Lite at 2K: 0.4036 versus 0.0988 ms ($4.085\times$); its sequential reference takes 0.1019 ms. These comparisons preserve both the large gains over particular overlap settings and the alternative of conventional execution.

F Historical H200 Training

The historical training study uses eight H200s, TP=4, EP=2, and synthetic input data. The DeepSeek-V3-derived workload has 64 experts, compared with 256 in the replay. The Llama4-derived workload has 32 experts and top-$k = 2$, compared with 128 and top-$k = 1$. Qwen1.5 and Qwen2 use 60/top-4 and 64/top-8, respectively. Depth is two or four layers as shown below.

Table 8: **Linear-projection comparison with official Transformer Engine.** HiPipe speedups are geometrically averaged over five token counts per workload; values above one favor HiPipe. All compared paths use $N = D_s/p$ within this experiment, whereas the primary gate-and-up experiments use $N = 2D_s/p$. The sequential reference performs no pipelining.

Workload	vs. sequential	vs. TE-SM	vs. TE-CE
Qwen1.5-MoE	1.010	2.284	2.073
Qwen2-57B	1.244	1.524	1.525
Hunyuan-Large	1.222	1.570	1.221
Llama4-Maverick	0.897	1.460	1.225
DeepSeek-V4-Pro	0.811	1.395	1.033
MiniMax-M3	0.842	1.484	1.189
DeepSeek-V2-Lite	0.975	1.894	1.877
DeepSeek-V3	0.872	1.564	1.137
DeepSeek-V4-Flash	0.855	1.558	1.404
GLM-5.2	0.766	1.444	1.136
Hy4-preview	0.760	1.405	1.119
Geometric mean	0.920	1.581	1.325

Table 9: **Reduced-model H200 training-step latency.** Entries are cleaned within-run means $\pm$ standard deviations (ms). Speedup is Megatron-overlap/HiPipe latency. Dashes denote unavailable measurements; the two missing 32K configurations are recorded as failed or out-of-memory runs. Standard deviations describe steps within a run, not uncertainty across independent launches.

Configuration	Layers	Sequential	Megatron OL	HiPipe	Speedup
DS-V3-8K	4	343.4 ± 26.8	339.5 ± 34.1	341.4 ± 20.8	0.995
DS-V3-16K	2	515.0 ± 32.3	506.1 ± 40.7	526.1 ± 30.1	0.962
DS-V3-32K	2	—	—	—	—
Llama4-8K	4	246.2 ± 12.8	241.1 ± 12.8	244.4 ± 11.7	0.987
Llama4-16K	2	305.7 ± 10.0	315.1 ± 24.6	314.0 ± 14.8	1.003
Llama4-32K	2	—	—	—	—
Qwen1.5-4K	4	163.4 ± 23.2	160.8 ± 20.0	158.3 ± 21.5	1.016
Qwen1.5-8K	4	255.3 ± 21.8	244.5 ± 27.4	253.5 ± 32.7	0.965
Qwen1.5-16K	4	390.7 ± 31.4	395.0 ± 41.2	386.2 ± 33.8	1.023
Qwen1.5-32K	2	483.1 ± 37.5	493.6 ± 28.2	478.9 ± 28.3	1.031
Qwen2-57B-8K	4	508.3 ± 43.2	499.2 ± 44.4	487.3 ± 42.1	1.024
Qwen2-57B-16K	4	675.7 ± 25.7	672.0 ± 32.5	661.8 ± 32.8	1.015
Qwen2-57B-32K	2	641.8 ± 42.8	622.9 ± 15.7	673.0 ± 49.8	0.926

The launcher specifies fifteen warm-up steps and twenty logged measurement steps. The existing cleaned file removes the first five logged steps and applies a $2\times$ interquartile-range filter. Most rows retain fifteen steps; some retain fewer. We preserve that summary without adding another filter. Independent paired runs should report raw and cleaned results, because filtering can affect differences of a few percent. Changes in depth also mean these rows are not a controlled sequence-scaling curve.

For upstream projection gradient $\overline{Z}_i$, the intended backward is

$$\overline{X}_i = \mathrm{RS}(\overline{Z}_i W_{1,i}^{\mathsf{T}}), \qquad \overline{W}_{1,i} = \mathrm{AG}(X_i)^{\mathsf{T}} \overline{Z}_i. \tag{3}$$

The wrapper expresses these operations using matrix multiplications and NCCL collectives. Establishing the integrated training path also requires matched gradients, current weights after updates, correct optimizer destinations, and activation lifetimes. A short paired loss/update trajectory should accompany the final step-time comparison.

If a fraction f of an otherwise unchanged step is accelerated by s, its idealized total speedup is $1/[(1-f) + f/s]$. No matched trace currently establishes f here. The expression explains why forward-operator gains can be diluted without predicting a numerical training gain.

G HARDWARE AND MEMORY

Table 10: **Layer-replay speedups over three reporting ranges.** The common range is 2K–32K; the main figure adds A800 64K; the full range also includes A800 512 and 1K. H200 retains the same 55 observations in every column. Ratios are geometric means over configurations, each normalized to its concurrent NCCL reference.

GPU	Common range		Main figure		Full range	
	Cases	Speedup	Cases	Speedup	Cases	Speedup
H200	55	0.997	55	0.997	55	0.997
A800	55	1.005	66	1.006	88	0.972

Table 11: **A800 shared-operator peak allocated memory at 32K tokens.** Values are in MiB (2^{20} bytes). Additional memory is the difference between recorded peaks. These measurements cover the shared operator, not a full model.

Workload at 32K	Baseline	Pipeline	Additional
Qwen1.5-MoE	393.5	577.6	184.1
Qwen2-57B	899.3	1215.4	316.1
Hunyuan-Large	1293.2	1851.3	558.1
Llama4-Maverick	935.5	1383.6	448.1
DeepSeek-V4-Pro	1137.6	1761.8	624.1
MiniMax-M3	1015.6	1543.6	528.0
DeepSeek-V2-Lite	458.3	634.3	176.0
DeepSeek-V3	1132.6	1748.6	616.0
DeepSeek-V4-Flash	720.1	1072.1	352.0
GLM-5.2	995.1	1523.1	528.0
Hy4-preview	995.1	1523.1	528.0

For element size b, a full exported input region occupies bSH bytes. With n_b staging buffers and r shards per group, staging occupies approximately $n_b\, b\, r(S/p)H$ bytes. The persistent output occupies bSN bytes; the baseline also needs a projection output, so the whole output is not automatically additional storage. Actual peak differences depend on allocation lifetimes and retained training activations. Reusing storage must preserve the ownership contract in Appendix A.

Complete shared-path replay. A separate historical A800 batch contains 77 configurations: eleven shapes at seven lengths from 512 to 32K. The interval includes AG/projection, a sliced projection output feeding FC2, and reduce-scatter, without routed work. Table 12 reports every length. The geometric mean changes from $0.750\times$ at 512 to $1.220\times$ at 32K. This shared-path scope is separate from input-only and layer timing. Its component measurements were collected in isolation: their sum and the recorded pipeline residual do not establish an event-level trace or a constant publication cost.

G.1 ADDITIONAL ROUTED-COMMUNICATION WORKLOADS

Two further archived batches contain 132 H200 and 264 A800 records. The harness fixes TP at eight and inserts two `all_to_all_single` calls around the routed matrix multiplications for stored tags 2 and 4; tag 1 omits those calls. Both communication tags use the same default eight-rank process group. They are retained as separately recorded scenarios, not distinct expert-parallel degrees or independent repetitions. H200 covers 2K, 8K, 16K, and 32K; A800 covers 512–64K.

Against the concurrent reference, H200's mean is $0.994\times$ without these calls and 1.028–$1.036\times$ in the two stored communication scenarios. A800 records $0.978\times$ and 1.015–$1.021\times$, respectively. The surrounding communication workload thus changes the observed relative performance. These replay summaries do not isolate a resource mechanism or establish a native-MoE expert-parallel scaling result.

Table 12: **A800 shared-path replay across input lengths.** Geometric-mean speedups across eleven shapes relative to sequential AG–FC1–FC2–RS; values above one favor HiPipe. The last column counts ratios above one. These 77 historical records have the execution-contract qualification described in Appendix A.

Tokens	Shapes	Speedup	> 1
512	11	0.750	0
1K	11	0.986	6
2K	11	1.122	9
4K	11	1.183	11
8K	11	1.192	11
16K	11	1.188	11
32K	11	1.220	11

Table 13: **Layer replays with additional routed communication.** Each row reports geometric-mean HiPipe speedups over all cases in one hardware/scenario group, using its concurrent or sequential reference. The last column counts concurrent-reference ratios above one. Tags retain the stored scenario identifiers, not actual expert-parallel group sizes.

GPU	Tag	A2A calls	Cases	vs. Concurrent	vs. Sequential	> 1
H200	1	0	44	0.994	1.077	16
H200	2	2	44	1.028	1.026	39
H200	4	2	44	1.036	1.030	40
A800	1	0	88	0.978	1.065	38
A800	2	2	88	1.021	1.014	67
A800	4	2	88	1.015	1.003	68

G.2 Captured input-stage overlap

Six historical H200 rank-zero traces cover Llama4- and DeepSeek-V3-derived shapes under sequential, NCCL-concurrent, and CE-concurrent execution. Their filenames contain `32K`, but the recorded matrix dimensions show 32K local rows and 256K gathered rows. They therefore describe a different workload from the global-32K measurements in the primary tables. We link GPU kernels to CPU matrix-multiplication records by their external identifiers and dimensions to identify the first projection.

The events demonstrate simultaneous peer transfer and projection, the execution structure used by HiPipe. The excerpt omits the subsequent shared output stage and routed completion, so its overlap cannot establish the effect on layer latency. The supplementary A800 event summaries in Appendix J.2 compare M2 and HiPipe on matching configurations, but do not provide the stage timeline needed to attribute the current native integration's costs.

H Completing the System Comparison

The remaining evaluation is organized around the claim it can establish. First, freeze a numerically equivalent implementation and matched baselines. Then evaluate the local primitive and the choice made for the enclosing schedule separately. This prevents an in-sample search result from being reported as an adaptive system result.

The existing full-width input controls and native-layer measurements appear in Tables 2 and 17. Additional matched evidence is needed for a full-width official-TE comparison, attribution of native HiPipe execution costs, and matched four-layer training gradients and updates. The supplementary checks in Appendix J.2 establish shared-parameter agreement for four single-layer configurations, while reporting input-gradient differences and the limits of the forward event timings. No constant correction is applied to historical timings to substitute for measuring these dependencies.

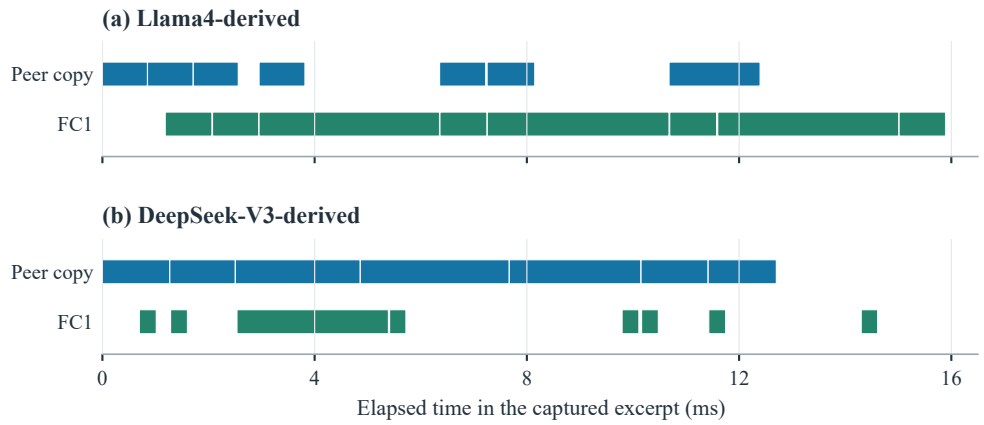


Figure 8: **Recorded transfer–projection overlap on H200.** Rank-zero excerpts for (a) Llama4-derived and (b) DeepSeek-V3-derived workloads, each containing the first eight FC1 kernels and seven peer copies. Bars preserve measured GPU-event intervals. Inputs contain 32K rows per rank and 256K after gathering. These profiled excerpts show execution structure, not unprofiled layer speedups.

H.1 Offline selection on held-out launches

The repeated A800 coverage permits retrospective evaluation of the choice criterion without rerunning benchmarks. For each of the 55 workloads, four launches select the candidate with the smallest geometric-mean latency under either the input or joint objective. The excluded launch supplies the selected candidate's joint latency and its same-launch NCCL reference. We rotate the excluded launch through all five possibilities. The candidate set is either the three measured groupings or those groupings plus conventional NCCL. The criterion never reads the excluded launch during selection.

Table 14: **Offline selection evaluated on held-out launches.** Four launches select a configuration per workload; the fifth evaluates joint latency against NCCL. Values geometrically average all 55 workloads and five rotations. Ranges span rotation-level means, not confidence intervals. Rotations share selection data, and calibration cost is excluded. Table 15 gives each rotation.

Criterion	NCCL option	Joint speedup	Fold range
Fixed $r = 1$	—	0.977	[0.971, 0.982]
Input	No	0.981	[0.976, 0.988]
Joint	No	0.985	[0.980, 0.991]
Input	Yes	1.003	[1.001, 1.006]
Joint	Yes	1.018	[1.016, 1.019]

Table 15: **Results for each held-out launch.** Entries are geometric-mean joint speedups over NCCL across 55 workloads. Fixed uses $r = 1$; Input and Joint select from $r \in \{1, 2, 4\}$ using their respective latency objectives. The final two columns also allow NCCL. Each row excludes its test launch during selection.

Test launch	Fixed	Input	Joint	Input+NCCL	Joint+NCCL
1	0.9749	0.9793	0.9835	1.0031	1.0178
2	0.9707	0.9760	0.9803	1.0013	1.0165
3	0.9770	0.9804	0.9835	1.0033	1.0181
4	0.9823	0.9879	0.9913	1.0060	1.0186
5	0.9787	0.9828	0.9871	1.0037	1.0179

The joint criterion selects conventional NCCL in 28–29 workloads per rotation, compared with 13 under the input criterion. This explains why changing the decision objective affects both grouping and whether pipelining is used. Table 14 averages the five fold-level means geometrically. The rotations share selection data, so their ranges are not confidence intervals from independent experiments. The analysis measures choices among recorded times, with no online profiling, calibration, switching, or amortization costs. It does not validate the prospective rule below, a deployed selector, or transfer from this proxy workload to native training.

H.2 Proposed calibration rule

Equation 9 motivates comparing configurations at joint completion; the local accounting in Appendix I.2 can help prioritize candidates. For a fixed workload, let $\mathcal{K}$ contain a small set of pipeline configurations and let $k^* = \arg\min_{k\in\mathcal{K}} \widehat{T}_{\text{join}}(k)$. A proposed enablement rule is

$$\widehat{T}_{\text{join}}(\text{ref}) - \widehat{T}_{\text{join}}(k^*) > 2\eta + T_{\text{cal}}/R, \tag{4}$$

where η bounds each latency estimate's uncertainty, T_{cal} is calibration time, and R is the expected number of repetitions. The conventional path is the fallback. The estimated saving must cover both uncertainty and amortized calibration.

This is a proposed selection rule. The offline analysis in Appendix H.1 tests the decision objective on excluded launches, but not this rule's uncertainty margin or calibration costs. Prospective evaluation of the rule remains to be completed. Its decisions must be frozen before the measurement runs. An in-sample best configuration describes the search space and does not establish the performance of a deployed selector. Appendix I.5 explains the uncertainty margin.

H.3 Comparison protocol

Does selection improve the schedule? Use separate calibration and measurement runs for fixed-off, fixed-on, isolated-operator selection, and joint selection. Freeze decisions before measuring held-out joint and training latency. Report calibration cost, the repetition count used to amortize it, regressions, and decision stability. An oracle chosen from the evaluation set can be an explicitly labeled upper bound, never the implemented selector.

Which offload boundary is useful? Compare matched AG-only and AG-plus-RS designs with disjoint initialized buffers, the same local work, and the same surrounding routed traffic. Record individual-operation durations, joint completion, and traces that distinguish scheduling from resource competition. Such an extension would require matched traces and joint timing for both offload boundaries. The current experiments do not establish an optimal AG-versus-RS offload boundary.

The archived stress test reports $100(T_{\text{joint}}/T_{\text{CE alone}} - 1)$ as "interference." That ratio compares two operations with one and is not a percentage loss of link bandwidth. Its inspected call site also uses `x_local` as both a concurrent RS destination and a CE input. Input ownership must be separated before attributing a contention mechanism. We therefore reserve causal claims for the matched experiment above.

I Derivations and Implementation Details

I.1 Input-pipeline implementation

Full shared computation. Let $N = 2D_s/p$ denote the local gate-and-up width. GPU i holds $W_{1,i} \in \mathbb{R}^{H\times N}$ and $W_{2,i} \in \mathbb{R}^{(D_s/p)\times H}$. All-gather assembles the full input X from the token shards X_i. The intermediate U_i contains this GPU's output contributions for all tokens, and V_i is its token shard after those contributions are summed:

$$Z_i = XW_{1,i}, \qquad U_i = \text{SwiGLU}(Z_i)W_{2,i}, \qquad V_i = \text{RS}(U_i). \tag{5}$$

SwiGLU reduces the width from $2D_s/p$ to D_s/p. Any native shared-expert gate is applied before the shared output is added to the routed output; V_i above is the MLP output before that gate.

Transfer and projection order. After projecting its local token shard, rank i visits peers in a ring order and uses asynchronous device-to-device copies through CUDA inter-process communication (IPC) mappings to stage their shards. Each transfer reads the mapped peer directly; the circular visit order does not imply multi-hop forwarding. The extension coordinates copies and ATen matrix multiplications through one C++ entry point. Copy engines execute the transfers, and ordinary matrix-multiplication kernels process the ready inputs.

The implementation combines grouping with two to four staging buffers and configurable copy and compute streams. Groups occupying a contiguous output range use one matrix multiplication; groups crossing the token-order boundary use the required sub-operations. These choices define the configuration space reported in Appendix D. All copies, projections, and buffer reuse must follow the ownership contract in Appendix A.

I.2 Local input-stage accounting

For conventional all-gather time A and unsplit projection time G, the input stage takes $A + G$. A pipeline configuration k introduces copy work C_k, partitioned projection work G_k, and fill, drain, and scheduling cost ϵ_k. A two-stage approximation is

$$T_{\text{pipe}}(k) \approx \max\{C_k, G_k\} + \epsilon_k, \qquad B_{\text{local}}(k) = A + G - T_{\text{pipe}}(k). \tag{6}$$

Splitting the projection can increase G_k relative to G even when more communication is overlapped. Grouping trades some early input consumption for larger matrix multiplications.

Expanding Equation 6 gives

$$B_{\text{local}}(k) \approx (A - C_k) - (G_k - G) + \min(C_k, G_k) - \epsilon_k. \tag{7}$$

The terms distinguish a communication-backend change, compute-efficiency change, hidden work, and scheduling cost. Attributing the whole observed difference to hidden communication would merge these effects.

I.3 Shared-expert width and transfer volume

For $p > 1$, the first projection performs $2SHN$ FLOPs, while all-gather receives $bSH(p-1)/p$ bytes per rank for element size b. Counting received payload once and using $N = 2D_s/p$ gives

$$\mathcal{I}_{\text{AG}} = \frac{2SHN}{bSH(p-1)/p} = \frac{4D_s}{b(p-1)}. \tag{8}$$

A wider shared expert offers more computation per transferred byte. Token count still matters through matrix shape and launch overhead even though S cancels from this ratio. Effective throughput, bandwidth, and the concurrent schedule determine the actual latency.

I.4 Relating shared savings to joint latency

Let T_{shared} and T_{routed} be the two branch latencies measured separately under the reference schedule. Consider a pipeline that reduces the isolated shared latency by δ, leaving the isolated routed latency unchanged. For either schedule, define the concurrency residual I as its measured joint latency minus the maximum of its two isolated branch latencies. Subscripts ref and pipe identify the conventional and pipelined schedules. Their measured difference is

$$\begin{aligned}\Delta T_{\text{joint}} = {} & \max(T_{\text{shared}}, T_{\text{routed}}) - \max(T_{\text{shared}} - \delta, T_{\text{routed}}) \\ & + (I_{\text{ref}} - I_{\text{pipe}}),\end{aligned} \tag{9}$$

where ΔT_{joint} is reference joint latency minus pipeline joint latency, so positive values denote savings. The difference between the two maxima is the saving that would remain if concurrency did not change the branch timings. The residual difference accounts for the remaining change in measured joint latency. It can include scheduling, synchronization, and resource effects, without identifying a hardware cause. This is an accounting of observed times, not a predictor of concurrent execution.

For $\delta \geq 0$, the difference between the maxima reduces to

$$B_{\text{exposed}} = \min\{\delta, \max(T_{\text{shared}} - T_{\text{routed}}, 0)\}. \tag{10}$$

If shared work already finishes sooner than routed work, this contribution is zero. Otherwise, it is the shared saving up to the original gap between the two isolated branch times. The residual term in Equation 9 is still needed to describe the measured joint saving.

I.5 Calibration margin

If every timing estimate deviates from its true value by at most η, the estimated reference-minus-candidate saving can overstate the true saving by at most 2η. Equation 4 additionally charges T_{cal}/R per repetition. Applying this guarantee to a selected candidate requires a bound valid for all candidates considered; pointwise uncertainty without selection adjustment is insufficient.

J A800 Protocols and Complete Results

The repeated A800 campaign is kept separate from the earlier replay campaigns. The operator and native-layer runs use eight A800-SXM4-80GB GPUs, PyTorch 2.4.0 with CUDA 12.1, BF16 inputs, and disabled TF32 and BF16 reduced-precision reduction. Model names denote architecture-derived shapes; the native inputs and weights are synthetic.

Aggregation and source records. Operator and native-layer timing records contain 200 samples after 50 warm-up iterations. For each sample we use the maximum across ranks, then take its p50 within a launch. Tables report medians of these p50s across launches. For paired comparisons, we geometrically average the within-launch baseline/candidate ratios. This generally differs from dividing the two displayed median latencies. An observed run range is not a confidence interval. The source manifest records each input file, its hash, and the individual values behind every aggregate. The operator batch shuffles method order; the native harness uses fixed method order, which repeated launches do not remove as a possible confound.

Recorded environment. Table 16 transcribes the available September 24 A800 environment snapshot. It documents that capture, not a recovered per-launch manifest for every campaign. Exact Megatron and kernel revisions, the historical H200 software stack, and the TE experiment version are not recoverable from this snapshot. The supplementary JSONs independently record A800-SXM4-80GB, TP8/EP1, and their UTC times.

Table 16: **Available A800 environment snapshot.** Software versions and topology are recorded values; configuration flags are read from the supplementary harness. Snapshot clocks are not treated as locked benchmark clocks.

Item	Recorded value
GPU array	8 × A800-SXM4-80GB
GPU links	NV8 between every recorded GPU pair
Driver	535.129.03
PyTorch	2.4.0+cu121
CUDA runtime / compiler	12.1 / 12.1.105
NCCL	2.20.5
Data type / TP / EP	BF16 / 8 / 1 (supplement)
CUDA device max connections	1 (supplement harness)
TF32 / BF16 reduced-precision reduction	Both disabled (supplement harness)

J.1 Native-layer coverage

The repeated batch contains 60 JSON records: five launches for each of three families at 4K, 8K, 16K, and 32K, all at TP8/EP1. The earlier pilot contains 16 records. When a configuration has a newer record, we use that cohort without pooling in its earlier pilot. Together the cohorts cover 20 configurations: 18 of the original 20 targets and the additional Qwen2 and DeepSeek-V3 4K/TP8 cases. Qwen1.5 and Qwen2 at 16K/TP4EP2 remain unmeasured in this snapshot and blank in Table 17.

The four forward-check failures are Qwen2 TP8/8K and DeepSeek-V3 TP8/8K, TP8/16K, and TP4/8K. Their recorded maximum relative-L2 errors range from 7.6×10^{-5} to 1.2×10^{-3}. We retain the stored pass/fail decisions rather than replacing the original numerical criterion after observing the results. All four Qwen1.5 TP8 configurations pass their matching receipts. The two added 4K/TP8 cases, Qwen2 and DeepSeek-V3, have no matching native forward receipt in the saved data.

Table 17: **Available A800 native-layer measurements.** Latencies are medians of launch-level p50s (ms); n is the launch count in the selected cohort. Speedup$_1$ is the geometric mean of paired Overlap/HiPipe ($r = 1$) ratios. †The stored forward check failed its criterion; ‡no matching forward check is available. Marked entries do not establish validated speedups; dashes denote missing measurements.

Tokens	TP/EP	n	Overlap	$r = 1$	$r = 2$	$r = 4$	Speedup$_1$
Qwen1.5-derived							
4K	8/1	5	12.553	13.096	12.964	13.098	0.980
4K	4/2	1	7.753	7.914	7.952	7.996	0.980
8K	8/1	5	12.979	13.205	13.271	13.296	0.984
8K	4/2	1	8.559	8.837	8.897	8.997	0.968
16K	8/1	5	14.493	14.948	15.128	15.174	0.972
16K	4/2	0	—	—	—	—	—
32K	8/1	5	18.477	19.108	19.269	19.508	0.967
32K	4/2	1	19.018	19.020	19.269	19.599	1.000
Qwen2-derived							
4K‡	8/1	5	14.694	15.454	15.459	15.463	0.958
8K†	8/1	5	17.762	18.925	18.663	18.785	0.933
8K	4/2	1	20.411	20.659	20.660	20.651	0.988
16K	8/1	5	31.509	31.473	31.530	31.759	1.001
16K	4/2	0	—	—	—	—	—
32K	8/1	5	58.726	58.703	58.723	59.139	1.000
32K	4/2	1	72.795	72.907	72.924	72.861	0.998
DeepSeek-V3-derived							
4K‡	8/1	5	48.312	48.472	48.711	48.870	0.987
8K†	8/1	5	52.902	54.044	53.994	53.857	0.981
8K†	4/2	1	35.673	36.177	36.588	36.802	0.986
16K†	8/1	5	65.587	67.005	67.303	67.122	0.982
16K	4/2	1	66.569	67.637	68.335	68.856	0.984
32K	8/1	5	109.648	110.965	112.126	112.204	0.988
32K	4/2	1	121.023	122.952	124.353	125.211	0.984

J.2 Supplementary backward checks and event summaries

The supplementary campaigns contain four matched single-layer backward records (E0-B v2) and four M2/HiPipe forward-timing records. All use TP8/EP1, BF16, synthetic inputs, and fixed $r = 1$ on eight A800 GPUs. These records are distinct from both the five-launch native timing cohort and the four-layer training stack.

Backward and optimizer checks. Within one distributed launch per configuration, serial, native overlap, and HiPipe layers load the same initial state and receive identical inputs. The harness checks initial parameter equality on each rank; the JSON saves the rank-zero equality flag. It clones outputs and gradients before comparing them elementwise. Four forward/backward checks use a sum loss, followed by a separate five-step Adam check at learning rate 10^{-4} from the initial state. Statistics cover all eight ranks and every step.

For HiPipe versus overlap, the four configurations yield 128 rank–step gradient observations. Each contains three shared-parameter tensors: the gate, first projection, and second projection. All 384 comparisons have zero maximum absolute error, relative L_2 error, and tolerance failure rate. The minimum reported cosine similarity is 0.99999982; cosines are computed in FP32 and need not evaluate to exactly one even for identical tensors. All 480 shared-update and 480 post-update shared-weight comparisons also have zero error; each compared update has nonzero norm. Table 18 summarizes these stored-tensor results. They replace the earlier unmatched rank-zero norm checks, which remain archived for provenance.

Table 18: **Matched single-layer shared-parameter checks.** HiPipe versus native overlap over all eight ranks. Errors are maximum absolute elementwise differences for gradients (four steps), updates, and post-update weights (five Adam steps). The cosine column gives the minimum shared-gradient cosine. Loss counts cover 40 rank-local step scalars per configuration.

Workload	Grad. error	Min. cosine	Update error	Weight error	Equal losses
Qwen1.5/4K	0	0.99999982	0	0	38/40
Qwen1.5/16K	0	0.99999988	0	0	39/40
Qwen2/8K	0	0.99999988	0	0	40/40
Qwen2/16K	0	0.99999988	0	0	40/40

This equality is specific to the shared parameters. Output and input-gradient differences remain (Table 19). The largest HiPipe–overlap input-gradient relative L_2 error is 5.99×10^{-4}, comparable in scale to overlap–serial error. Some elements exceed the recorded tolerance criterion, so these records do not establish a full-layer tolerance pass. During Adam checks, 157 of 160 rank-local losses match exactly; three Qwen1.5 scalars differ by 4.0, 0.25, and 0.125. All 20 rank-zero losses match, but this subset is not the all-rank result.

Table 19: **Output and input-gradient differences.** H/O denotes HiPipe versus overlap; O/S denotes overlap versus serial. The first three numerical columns are relative L_2 errors. The final two give percentages of elements exceeding both $|a-b| > 10^{-3}$ and $|a-b|/\max(|b|, 10^{-8}) > 10^{-2}$ for H/O. Each entry is the maximum over four steps and eight ranks.

Workload	Y: H/O	dX: H/O	dX: O/S	Y (%)	dX (%)
Qwen1.5/4K	3.97×10^{-4}	5.99×10^{-4}	5.89×10^{-4}	0.03700	0.22049
Qwen1.5/16K	7.69×10^{-5}	3.34×10^{-4}	3.39×10^{-4}	0.00124	0.06995
Qwen2/8K	6.48×10^{-5}	4.37×10^{-4}	4.41×10^{-4}	0.00550	0.33493
Qwen2/16K	0	3.74×10^{-4}	3.87×10^{-4}	0.00000	0.25759

The checks support shared-parameter gradient and update agreement for these four single-layer, sum-loss workloads. They do not compare Adam moment tensors, cover arbitrary upstream gradients, or validate the separate four-layer training stack. Loading matched states removes an initialization confound in the earlier checks without identifying the exclusive cause of their historical discrepancies.

Forward event summaries. For each configuration, the harness measures M2 followed by HiPipe on the same layer and eight-input pool, with 30 warm-up and 100 measured iterations per method. It takes the rank maximum per sample and reports its p50 and p95. The ratio in Table 20 divides the two p50s from this single launch; it is not pooled with the five-launch geometric-mean ratios in Table 17. The results agree in direction with that cohort. Qwen2/16K differs by only 0.029 ms, approximately 0.09%, and is treated as near parity.

Table 20: **Supplementary A800 forward event summaries.** Layer and marker times are milliseconds from each rank's layer-start event, aggregated by rank maximum and sample p50. Ratio is M2/HiPipe layer p50. The shared-stream marker is recorded at `get_output` entry; it is not a validated shared-output completion timestamp.

Workload	Layer M2	Layer HiPipe	Ratio	Marker M2	Marker HiPipe
Qwen1.5/4K	12.426	12.586	0.9873	12.305	12.466
Qwen1.5/16K	14.553	14.923	0.9752	14.302	14.671
Qwen2/8K	17.786	18.950	0.9386	17.375	18.540
Qwen2/16K	31.500	31.471	1.0009	30.649	30.621

The shared marker is enqueued only when the host reaches `get_output`, before calling its original implementation. A late host call can therefore delay the marker even if earlier shared work has finished; any work inside that implementation is also outside the marker's scope. Further, the maximum layer and marker times can come from different ranks. Their difference is not a measured per-rank slack. Without routed completion, join waits, and stage intervals, these summaries cannot identify the limiting branch or isolate publication cost. The archived JSONs contain these aggregates, not per-sample or raw profiler timelines. They support the layer-timing comparison while leaving the mechanism attribution open.

Scope of the observed boundary. This supplement samples only $D_s = 5632$ and 20480, while hidden width and routed workloads also differ. It does not establish a fixed shared-width threshold. Indeed, the repeated input/replay campaign records Qwen1.5 with $D_s = 5632$ at 32K improving by $1.549\times$ and $1.057\times$ in those two scopes (Table 1). Input length, grouping, and the surrounding execution must accompany any width-based interpretation.

J.3 Input-stage controls and coverage

The six-workload backend-control batch uses the same five launches for NCCL, CE gather-then-project, and HiPipe with $r \in \{1, 2, 4\}$. Its joint replay adds shared SwiGLU, FC2, and reduce-scatter alongside a dense routed proxy; it has no actual router or expert all-to-all. The full-width input stage uses $N = 2D_s/p$ for every method.

Table 21: **A800 backend-control speedups with $r = 1$.** Geometric means of five NCCL/HiPipe ratios, paired within the same scope and launch. Brackets report the observed minimum and maximum, not confidence intervals. Input projection and synthetic joint replay are separately timed scopes.

Shape	Tokens	Input	Run range	Joint	Run range
Qwen1.5	8K	1.142	[1.092, 1.241]	1.040	[1.030, 1.049]
Qwen1.5	32K	1.546	[1.536, 1.556]	1.159	[1.156, 1.163]
Qwen2	8K	0.976	[0.975, 0.977]	0.990	[0.988, 0.991]
Qwen2	32K	1.227	[1.225, 1.229]	1.067	[1.065, 1.070]
DeepSeek-V3	8K	1.348	[1.332, 1.375]	1.069	[1.068, 1.072]
DeepSeek-V3	32K	1.420	[1.417, 1.428]	1.065	[1.064, 1.067]

A separate coverage batch measures all eleven shapes at 2K, 4K, 8K, 16K, and 32K, with five launches. Table 22 first averages ratios geometrically across the five lengths within each launch, then across the five launches. The original six-workload batch, the backend-control batch, and this coverage batch remain separate; we do not select the most favorable value across them.

Numerical-check provenance. The five E1-C timing records specify timed current-input publication, but do not contain a complete matching numerical-check receipt. Two earlier current-input check artifacts cover Qwen1.5 and Qwen2 at 8K/32K with passing records, then stop at a DeepSeek-V3/8K failure: input projection in the default-BF16 attempt and joint replay in the FP32-accumulation attempt. These earlier receipts do not validate the full eleven-shape, five-length timing cohort. A saved E1-C maintenance script also removes correctness-gate exceptions; its presence does not establish which timed revision used that change. Linking the executed revision to complete numerical receipts remains necessary. We retain the measured timing ratios without promoting process completion to numerical validation.

Table 22: **Complete eleven-shape A800 operator coverage.** Geometric-mean NCCL/HiPipe ratios at TP8, over five token counts and five launches per grouping. Values above one favor HiPipe. Synthetic joint replay uses a dense routed proxy. The source data preserve every input length and launch.

Shape	Input projection			Synthetic joint		
	$r = 1$	$r = 2$	$r = 4$	$r = 1$	$r = 2$	$r = 4$
Qwen1.5	1.076	0.937	0.846	0.952	0.935	0.898
Qwen2	1.029	1.059	0.954	0.980	0.991	0.972
Hunyuan	1.030	1.066	1.019	0.979	1.001	1.002
Llama4	1.131	1.101	0.978	0.985	0.995	0.988
DeepSeek-V2-Lite	0.965	0.834	0.774	0.929	0.927	0.908
DeepSeek-V3	1.270	0.976	0.919	0.981	0.977	0.975
DeepSeek-V4-Flash	1.122	0.906	0.855	0.985	0.963	0.971
DeepSeek-V4-Pro	1.295	0.998	0.914	1.014	1.002	0.995
GLM-5.2	1.221	0.949	0.898	0.972	0.977	0.971
Hy4-preview	1.209	0.948	0.894	0.971	0.975	0.971
MiniMax-M3	1.261	0.977	0.903	0.999	0.996	0.988

Table 23: **Grouping effects by input length on A800.** NCCL/HiPipe ratios are geometrically averaged first across eleven shapes within a launch, then across five launches. The table gives the points in Figure 4. Each grouping is fixed across workloads; no per-case selection enters these means.

Tokens	Input projection			Joint replay		
	$r = 1$	$r = 2$	$r = 4$	$r = 1$	$r = 2$	$r = 4$
2K	0.850	0.810	0.754	0.869	0.864	0.858
4K	1.023	0.914	0.856	0.978	0.968	0.943
8K	1.175	1.001	0.923	0.988	0.989	0.978
16K	1.337	1.081	0.985	1.022	1.024	1.018
32K	1.416	1.098	1.021	1.036	1.044	1.047

Grouping is evaluated under both completion objectives. Of 55 workloads, the input-stage point-estimate optimum uses $r = 1, 2, 4$ in 45, 9, and 1 cases, respectively. The joint-replay counts are 23, 17, and 15. The chosen grouping differs across scopes in 27 cases. These counts describe the largest geometric-mean ratio without a significance threshold. They do not establish that every small difference is distinguishable or estimate a selector's out-of-sample performance. At 32K, however, the aggregate ordering favoring $r = 1$ for input projection and $r = 4$ for joint completion holds in each of the five launch-level means.

The earlier official TE experiment in Appendix E uses $N = D_s/p$, rather than the full combined gate-and-up width here. Its numbers are therefore not inserted into the full-width comparison.

J.4 Training-step records and memory

The A800 training microbenchmark stacks four native MoE layers, with random hidden inputs, a sum loss, and Adam updates. It uses TP8/EP1, 20 warm-up steps, and 100 measured steps per launch. The saved step time is rank-zero wall time through forward, backward, and optimizer update after device

synchronization; it is not a saved rank-max distribution. Table 24 uses five launches per method and configuration.

Only revised HiPipe-v2 records are shown; the superseded training implementation is excluded. Its input-publication and backward paths were corrected. Matched single-layer backward and Adam checks show shared-gradient and shared-update agreement across all eight ranks (Appendix J.2); matched elementwise gradients and updates for this four-layer stack remain unverified. Baseline and HiPipe-v2 timings were collected in separate campaigns. We therefore report descriptive times without a validated training speedup.

Table 24: **Recorded A800 MoE-stack training-step latency.** Each cell gives the median of five launch-level p50s (ms), with their observed range below. All configurations use four MoE layers and TP8/EP1. †Four-layer HiPipe-v2 gradient/update equivalence and a paired training comparison remain to be established.

Shape	Tokens	Serial	Overlap	HiPipe-v2†
Qwen1.5	8K	141.6 [139.4,143.7]	140.2 [137.0,144.0]	141.8 [140.2,143.8]
Qwen1.5	16K	144.1 [141.0,146.1]	143.3 [141.6,144.3]	147.3 [146.0,149.2]
Qwen1.5	32K	191.9 [191.7,194.5]	177.9 [177.4,180.2]	190.6 [189.4,193.2]
Qwen2	8K	242.8 [228.5,250.2]	230.1 [228.1,235.8]	244.3 [239.3,251.1]
Qwen2	16K	351.6 [348.3,360.0]	327.8 [324.1,333.3]	342.8 [342.5,353.7]
Qwen2	32K	616.9 [616.3,617.7]	569.6 [566.5,572.3]	602.0 [601.7,603.1]

Table 25: **Peak allocated memory during A800 MoE-stack training.** Medians across five launches per method, in GiB (2^{30} bytes). Values are per-process implementation peaks, not isolated pipeline storage. †The training-equivalence qualification in Table 24 also applies.

Shape	Tokens	Serial	Overlap	HiPipe-v2†
Qwen1.5	8K	2.93	3.14	3.13
Qwen1.5	16K	4.20	4.54	4.46
Qwen1.5	32K	6.83	7.44	7.24
Qwen2	8K	10.16	10.55	10.62
Qwen2	16K	14.65	15.35	15.30
Qwen2	32K	23.59	24.94	24.67

K Complete Layer and Branch Replay Latencies

Tables 26 and 27 report every configuration behind the historical layer heatmaps: 55 on H200 and 88 on A800. Sequential and concurrent execution use the same workload as HiPipe within each campaign. The tables retain measured per-layer latencies; depth-multiplied total-time estimates are omitted. These synthetic replays use attention-shaped and dense routed matrix workloads, with the scope described in Appendix B. Native-layer results remain in Table 17.

The archived records contain rounded latency summaries and separately rounded speedup ratios. We preserve both, so recomputing a ratio from the displayed milliseconds can differ slightly from its saved value. The number of independent launches and run-level uncertainty are not recoverable from these summaries. Each row denotes a configuration, not an independent repetition. Rows follow the projection-width order of Figure 3; all measured token counts are retained.

Table 26: **Complete H200 layer-replay records.** All 55 configurations, TP8, 2K–32K. Latencies are in ms. Speedups divide the serial or concurrent NCCL latency by HiPipe; values above one favor HiPipe. Latencies and ratios retain their separately rounded source values. These are synthetic replays, not native-layer measurements.

Derived workload	Tokens	Latency (ms)			Speedup	
		Serial	Overlap	HiPipe	vs. Seq.	vs. Conc.
Qwen2-57B	2K	0.560	0.582	0.543	1.031	1.072
Qwen2-57B	4K	0.988	0.940	0.899	1.099	1.045
Qwen2-57B	8K	1.791	1.662	1.641	1.091	1.012
Qwen2-57B	16K	3.470	3.229	3.176	1.093	1.017
Qwen2-57B	32K	7.004	6.430	6.109	1.147	1.052
Hunyuan	2K	0.920	0.892	0.881	1.045	1.013
Hunyuan	4K	1.610	1.613	1.566	1.028	1.030
Hunyuan	8K	3.100	2.889	2.828	1.096	1.022
Hunyuan	16K	6.124	5.582	5.572	1.099	1.002
Hunyuan	32K	11.809	11.116	10.362	1.140	1.073
Llama4	2K	0.550	0.525	0.538	1.022	0.977
Llama4	4K	0.951	0.925	0.855	1.112	1.082
Llama4	8K	1.752	1.675	1.555	1.127	1.078
Llama4	16K	3.326	3.161	2.879	1.155	1.098
Llama4	32K	6.486	6.060	5.539	1.171	1.094
Qwen1.5	2K	0.263	0.255	0.264	0.998	0.968
Qwen1.5	4K	0.407	0.395	0.378	1.075	1.044
Qwen1.5	8K	0.713	0.697	0.625	1.141	1.115
Qwen1.5	16K	1.274	1.196	1.105	1.152	1.082
Qwen1.5	32K	2.368	2.261	2.131	1.111	1.061
DS-V4-Pro	2K	0.781	0.728	0.717	1.089	1.015
DS-V4-Pro	4K	1.350	1.232	1.270	1.063	0.970
DS-V4-Pro	8K	2.602	2.345	2.412	1.079	0.972
DS-V4-Pro	16K	5.159	4.496	4.687	1.101	0.959
DS-V4-Pro	32K	9.803	8.846	9.152	1.071	0.967
MiniMax-M3	2K	0.615	0.595	0.592	1.040	1.006
MiniMax-M3	4K	1.056	0.993	0.991	1.065	1.002
MiniMax-M3	8K	1.960	1.817	1.850	1.059	0.982
MiniMax-M3	16K	3.819	3.429	3.537	1.079	0.969
MiniMax-M3	32K	7.454	6.671	7.037	1.059	0.948

Continued on next page

Table 26 continued: H200 layer replays

Derived workload	Tokens	Latency (ms)			Speedup	
		Serial	Overlap	HiPipe	vs. Seq.	vs. Conc.
DS-V2-Lite	2K	0.260	0.248	0.258	1.010	0.963
DS-V2-Lite	4K	0.401	0.375	0.379	1.057	0.989
DS-V2-Lite	8K	0.684	0.657	0.687	0.995	0.955
DS-V2-Lite	16K	1.222	1.142	1.112	1.099	1.027
DS-V2-Lite	32K	2.284	2.117	2.116	1.079	1.000
DS-V3	2K	0.740	0.697	0.708	1.045	0.984
DS-V3	4K	1.283	1.191	1.234	1.040	0.965
DS-V3	8K	2.421	2.207	2.277	1.063	0.969
DS-V3	16K	4.803	4.227	4.489	1.070	0.942
DS-V3	32K	9.242	8.283	8.776	1.053	0.944
DS-V4-Flash	2K	0.422	0.399	0.417	1.010	0.956
DS-V4-Flash	4K	0.731	0.693	0.685	1.066	1.011
DS-V4-Flash	8K	1.301	1.204	1.230	1.058	0.979
DS-V4-Flash	16K	2.461	2.231	2.312	1.065	0.965
DS-V4-Flash	32K	4.739	4.218	4.401	1.077	0.959
GLM-5.2	2K	0.632	0.598	0.605	1.044	0.988
GLM-5.2	4K	1.089	1.003	1.036	1.052	0.968
GLM-5.2	8K	2.040	1.850	1.942	1.050	0.952
GLM-5.2	16K	4.070	3.558	3.806	1.069	0.935
GLM-5.2	32K	7.834	6.936	7.303	1.073	0.950
Hy4-preview	2K	0.634	0.595	0.601	1.055	0.991
Hy4-preview	4K	1.091	1.009	1.035	1.054	0.974
Hy4-preview	8K	2.039	1.847	1.948	1.047	0.948
Hy4-preview	16K	4.049	3.552	3.804	1.065	0.934
Hy4-preview	32K	7.993	6.935	7.422	1.077	0.934
All 55 cases (GM)					1.072	0.997

Table 27: **Complete A800 layer-replay records.** All 88 configurations, TP8, 512–64K. Latencies are in ms. Speedups divide the serial or concurrent NCCL latency by HiPipe; values above one favor HiPipe. Latencies and ratios retain their separately rounded source values. These are synthetic replays, not native-layer measurements.

Derived workload	Tokens	Latency (ms)			Speedup	
		Serial	Overlap	HiPipe	vs. Seq.	vs. Conc.
Qwen2-57B	512	0.543	0.504	0.607	0.894	0.830
Qwen2-57B	1K	0.946	0.765	0.868	1.089	0.881
Qwen2-57B	2K	1.557	1.399	1.441	1.081	0.971
Qwen2-57B	4K	2.571	2.372	2.467	1.042	0.961
Qwen2-57B	8K	4.796	4.404	4.655	1.030	0.946
Qwen2-57B	16K	9.115	8.506	8.310	1.097	1.024
Qwen2-57B	32K	18.057	17.041	16.185	1.116	1.053
Qwen2-57B	64K	35.834	34.178	32.158	1.114	1.063

Continued on next page

Table 27 continued: A800 layer replays

Derived workload	Tokens	Latency (ms)			Speedup	
		Serial	Overlap	HiPipe	vs. Seq.	vs. Conc.
Hunyuan	512	0.881	0.846	1.124	0.784	0.752
Hunyuan	1K	1.456	1.372	1.446	1.007	0.949
Hunyuan	2K	2.382	2.168	2.206	1.080	0.983
Hunyuan	4K	4.405	4.055	4.078	1.080	0.994
Hunyuan	8K	8.160	7.490	7.932	1.029	0.944
Hunyuan	16K	15.682	14.659	14.666	1.069	1.000
Hunyuan	32K	30.965	29.372	28.167	1.099	1.043
Hunyuan	64K	61.277	58.424	55.513	1.104	1.052
Llama4	512	0.602	0.544	0.681	0.884	0.800
Llama4	1K	0.963	0.914	0.931	1.034	0.982
Llama4	2K	1.568	1.442	1.391	1.127	1.036
Llama4	4K	2.655	2.364	2.327	1.141	1.016
Llama4	8K	4.481	4.119	4.043	1.108	1.019
Llama4	16K	8.533	7.819	7.990	1.068	0.979
Llama4	32K	16.505	15.253	14.804	1.115	1.030
Llama4	64K	32.399	30.052	28.293	1.145	1.062
Qwen1.5	512	0.348	0.387	0.522	0.666	0.740
Qwen1.5	1K	0.464	0.456	0.465	0.998	0.982
Qwen1.5	2K	0.695	0.620	0.690	1.007	0.898
Qwen1.5	4K	1.260	1.128	1.061	1.188	1.063
Qwen1.5	8K	1.938	1.796	1.705	1.137	1.053
Qwen1.5	16K	3.073	2.900	2.672	1.150	1.085
Qwen1.5	32K	5.837	5.463	5.082	1.149	1.075
Qwen1.5	64K	11.356	10.578	9.703	1.170	1.090
DS-V4-Pro	512	0.678	0.604	0.678	1.000	0.891
DS-V4-Pro	1K	1.183	1.047	1.057	1.119	0.990
DS-V4-Pro	2K	2.017	1.774	1.726	1.169	1.028
DS-V4-Pro	4K	3.604	3.213	3.204	1.125	1.003
DS-V4-Pro	8K	6.669	5.992	5.914	1.128	1.013
DS-V4-Pro	16K	12.875	11.698	11.727	1.098	0.998
DS-V4-Pro	32K	25.045	23.283	23.076	1.085	1.009
DS-V4-Pro	64K	49.482	47.296	46.534	1.063	1.016
MiniMax-M3	512	0.535	0.502	0.552	0.970	0.910
MiniMax-M3	1K	0.935	0.872	0.860	1.087	1.014
MiniMax-M3	2K	1.552	1.417	1.389	1.117	1.020
MiniMax-M3	4K	2.719	2.437	2.436	1.116	1.000
MiniMax-M3	8K	5.104	4.558	4.480	1.139	1.017
MiniMax-M3	16K	9.810	8.653	8.623	1.138	1.004
MiniMax-M3	32K	19.049	17.209	17.164	1.110	1.003
MiniMax-M3	64K	37.157	34.116	34.048	1.091	1.002
DS-V2-Lite	512	0.274	0.358	0.487	0.562	0.734
DS-V2-Lite	1K	0.373	0.358	0.480	0.778	0.746
DS-V2-Lite	2K	0.561	0.566	0.621	0.902	0.911
DS-V2-Lite	4K	1.023	0.938	0.899	1.138	1.044
DS-V2-Lite	8K	1.652	1.470	1.485	1.112	0.990
DS-V2-Lite	16K	2.926	2.670	2.581	1.134	1.034
DS-V2-Lite	32K	5.579	5.068	4.981	1.120	1.018
DS-V2-Lite	64K	10.837	9.745	9.706	1.117	1.004

Continued on next page

Table 27 continued: A800 layer replays

Derived workload	Tokens	Latency (ms)			Speedup	
		Serial	Overlap	HiPipe	vs. Seq.	vs. Conc.
DS-V3	512	0.689	0.581	0.644	1.070	0.902
DS-V3	1K	1.134	1.001	1.017	1.115	0.985
DS-V3	2K	1.869	1.687	1.654	1.130	1.020
DS-V3	4K	3.430	3.136	3.111	1.102	1.008
DS-V3	8K	6.254	5.595	5.697	1.098	0.982
DS-V3	16K	12.277	10.969	11.007	1.115	0.997
DS-V3	32K	23.546	21.834	22.121	1.064	0.987
DS-V3	64K	46.601	42.536	43.548	1.070	0.977
DS-V4-Flash	512	0.409	0.405	0.571	0.717	0.709
DS-V4-Flash	1K	0.614	0.551	0.610	1.007	0.903
DS-V4-Flash	2K	1.052	0.990	0.942	1.117	1.051
DS-V4-Flash	4K	1.770	1.541	1.564	1.131	0.985
DS-V4-Flash	8K	3.282	2.860	2.837	1.157	1.008
DS-V4-Flash	16K	6.250	5.528	5.527	1.131	1.000
DS-V4-Flash	32K	12.098	10.566	11.024	1.097	0.959
DS-V4-Flash	64K	23.464	20.780	21.692	1.082	0.958
GLM-5.2	512	0.546	0.496	0.539	1.013	0.920
GLM-5.2	1K	0.980	0.860	0.879	1.114	0.978
GLM-5.2	2K	1.718	1.519	1.456	1.179	1.043
GLM-5.2	4K	2.943	2.649	2.599	1.133	1.019
GLM-5.2	8K	5.435	4.794	4.865	1.117	0.986
GLM-5.2	16K	10.287	9.157	9.244	1.113	0.991
GLM-5.2	32K	19.883	18.054	18.488	1.075	0.977
GLM-5.2	64K	39.296	35.795	36.981	1.063	0.968
Hy4-preview	512	0.548	0.499	0.608	0.901	0.820
Hy4-preview	1K	0.989	0.888	0.885	1.118	1.004
Hy4-preview	2K	1.715	1.510	1.457	1.177	1.037
Hy4-preview	4K	2.935	2.622	2.603	1.127	1.007
Hy4-preview	8K	5.437	4.860	4.839	1.124	1.004
Hy4-preview	16K	10.312	9.169	9.254	1.114	0.991
Hy4-preview	32K	19.891	18.011	18.528	1.073	0.972
Hy4-preview	64K	39.292	35.604	36.930	1.064	0.964
All 88 cases (GM)					1.060	0.972

K.1 Paired shared and joint intervals

Table 28 contains all 55 H200 branch records. It retains the same shape order as the layer tables and reports shared-alone and joint times against their own NCCL references. The difference between an isolated shared saving and a joint saving is analyzed in Equation 9; it is not an observed duration of a particular hardware resource stall.

Table 28: **Complete H200 branch-replay times.** All 55 configurations, TP8. Shared-alone and joint latencies are in ms and timed separately. Joint speedup is concurrent-NCCL/HiPipe latency; values above one favor HiPipe. These are recorded point estimates, without independent-run intervals.

Workload	Tokens	Shared alone (ms)		Joint (ms)		Speedup
		NCCL	HiPipe	NCCL	HiPipe	Joint
Qwen2-57B	2K	0.290	0.268	0.376	0.384	0.979
Qwen2-57B	4K	0.519	0.461	0.694	0.637	1.090
Qwen2-57B	8K	0.944	0.746	1.235	1.212	1.018
Qwen2-57B	16K	1.830	1.483	2.441	2.472	0.988
Qwen2-57B	32K	3.586	2.927	5.064	4.741	1.068
Hunyuan	2K	0.451	0.405	0.631	0.615	1.026
Hunyuan	4K	0.804	0.686	1.122	1.065	1.054
Hunyuan	8K	1.561	1.258	2.083	2.102	0.991
Hunyuan	16K	3.040	2.499	4.179	4.211	0.993
Hunyuan	32K	6.320	4.632	8.373	7.391	1.133
Llama4	2K	0.259	0.259	0.321	0.286	1.124
Llama4	4K	0.450	0.370	0.500	0.443	1.129
Llama4	8K	0.831	0.647	0.896	0.706	1.269
Llama4	16K	1.588	1.216	1.898	1.314	1.444
Llama4	32K	3.083	2.297	3.288	2.509	1.310
Qwen1.5	2K	0.129	0.241	0.228	0.345	0.662
Qwen1.5	4K	0.197	0.187	0.524	0.206	2.544
Qwen1.5	8K	0.348	0.300	0.414	0.364	1.138
Qwen1.5	16K	0.609	0.517	0.690	0.607	1.137
Qwen1.5	32K	1.154	0.928	1.302	1.117	1.165
DS-V4-Pro	2K	0.281	0.254	0.434	0.421	1.032
DS-V4-Pro	4K	0.467	0.424	0.750	0.738	1.016
DS-V4-Pro	8K	0.867	0.780	1.371	1.416	0.968
DS-V4-Pro	16K	1.691	1.497	2.639	2.742	0.962
DS-V4-Pro	32K	3.386	2.727	5.230	5.434	0.963
MiniMax-M3	2K	0.247	0.230	0.339	0.339	0.998
MiniMax-M3	4K	0.407	0.369	0.571	0.541	1.056
MiniMax-M3	8K	0.746	0.671	1.011	1.000	1.011
MiniMax-M3	16K	1.430	1.264	1.854	1.955	0.949
MiniMax-M3	32K	2.829	2.433	3.626	3.813	0.951
DS-V2-Lite	2K	0.114	0.128	0.138	0.171	0.808
DS-V2-Lite	4K	0.174	0.175	0.212	0.215	0.989
DS-V2-Lite	8K	0.301	0.278	0.381	0.346	1.099
DS-V2-Lite	16K	0.528	0.467	0.654	0.594	1.100
DS-V2-Lite	32K	0.987	0.882	1.196	1.153	1.038
DS-V3	2K	0.263	0.254	0.387	0.402	0.961
DS-V3	4K	0.440	0.411	0.651	0.678	0.961
DS-V3	8K	0.809	0.747	1.181	1.263	0.935
DS-V3	16K	1.595	1.441	2.309	2.555	0.904
DS-V3	32K	3.146	2.630	4.535	4.933	0.919

Continued on next page

Table 28 continued: H200 branch replays

Workload	Tokens	Shared alone (ms)		Joint (ms)		Speedup
		NCCL	HiPipe	NCCL	HiPipe	Joint
DS-V4-Flash	2K	0.168	0.179	0.225	0.312	0.722
DS-V4-Flash	4K	0.285	0.271	0.390	0.479	0.816
DS-V4-Flash	8K	0.494	0.478	0.677	0.676	1.002
DS-V4-Flash	16K	0.919	0.847	1.221	1.261	0.968
DS-V4-Flash	32K	1.765	1.610	2.312	2.415	0.957
GLM-5.2	2K	0.234	0.223	0.341	0.348	0.978
GLM-5.2	4K	0.381	0.407	0.570	0.665	0.857
GLM-5.2	8K	0.698	0.654	1.030	1.086	0.948
GLM-5.2	16K	1.346	1.227	1.980	2.147	0.922
GLM-5.2	32K	2.680	2.392	3.873	4.206	0.921
Hy4-preview	2K	0.236	0.224	0.340	0.345	0.987
Hy4-preview	4K	0.383	0.361	0.570	0.586	0.971
Hy4-preview	8K	0.697	0.649	1.028	1.107	0.928
Hy4-preview	16K	1.341	1.235	2.024	2.224	0.910
Hy4-preview	32K	2.685	2.394	3.890	4.231	0.919

Artifact contents. The source package contains the LaTeX dependencies, vector figure assets, numerical table generators, historical H200/A800 summaries, and newer A800 per-launch aggregates. It also includes the four matched E0-B v2 JSONs and their harness, retaining the eight earlier supplementary JSONs and six historical gradient-reference records. Portable generators rebuild Tables 16, 18, 19, and 20. Source manifests link aggregates to their original files. The two schematics carry no measured time scale; the scatter retains all 55 branch records. Table 16 supplies the available A800 environment snapshot. Per-campaign revision matching, missing software versions, clock policy, and complete numerical receipts remain explicit reproducibility limits.