KV-PIPE: STAGE-AWARE KV SHARING FOR BALANCED PIPELINE-PARALLEL LLM TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Pipeline parallelism (PP) is bounded by its slowest stage. In LLMs, however, the stages are not identical: the last stage also carries the LM head, so an evenly split decoder already leaves every other device idle for part of each step. Moving whole layers between stages cannot fix this, because a single layer is coarser than the imbalance itself. We observe that cross-layer KV sharing, which has so far been studied as an inference-time memory optimization, is exactly the fine-grained knob this problem needs. Each shared layer removes a small, predictable amount of compute from its stage. We present KV-Pipe, which places shared-KV layers on the bottleneck stage and chooses how many to share by minimizing a FLOPs Imbalance Ratio (FIR) computed offline from per-layer FLOP counts. For LLaMA2-7B with PP=8 on Ascend 910B NPUs, sharing KV in only four layers removes 1.5% of model FLOPs but cuts iteration time by 9.8% and raises MFU from 56.3% to 61.5%. Across the sweep, FIR orders measured MFU almost perfectly (Spearman $\rho = -1.0$ at PP=8). Sharing more layers makes both FIR and MFU worse again, even though total FLOPs keep falling. When we vary the LM-head cost, the budget predicted from FIR matches the measured optimum in all five settings. The gains grow with pipeline depth, also in a controlled sweep that fixes everything but the number of stages. On LLaMA2-7B and LLaMA3-8B, validation perplexity and downstream accuracy stay within the run-to-run variation of full attention. The gains are additive to the Seq1F1B schedule, carry over to LLaMA2-13B, LLaMA3-8B and Qwen2.5-14B on GPUs, and the same kind of late-layer sharing speeds up GQA decoding by 7–8%.

1 INTRODUCTION

Training a large language model (LLM) requires splitting it across many accelerators, and pipeline parallelism (PP) (Huang et al., 2019; Narayanan et al., 2019; 2021) is the standard way to split it across depth. PP has cheap communication, but its throughput is set by its slowest stage. Every microsecond by which one stage exceeds the others is paid as idle time on all remaining devices. A large body of work reduces this idle time by better scheduling (1F1B, interleaving, zero-bubble and sequence-level schedules) (Narayanan et al., 2021; Qi et al., 2024; Ao et al., 2025), by searching for better layer partitions (Fan et al., 2020; Zhao et al., 2022; Peng et al., 2025), or by filling bubbles with other work (Arfeen et al., 2025). All of these methods take the model's per-layer cost profile as fixed.

For LLMs this assumption hides a structural problem. Decoder layers are identical, but the pipeline stages that hold them are not. The first stage also holds the embedding and the last stage holds the LM head and loss. For LLaMA2-7B at 8K context, the LM head costs as much as 0.56 decoder layers. With 32 layers split evenly over 8 stages, the last stage therefore carries 13.9% more work than the others (FIR $= 1.12$, Section 2). In steady state the other seven devices are idle for roughly 11% of every step (Figure 1). Layer-level repartitioning cannot remove this surplus. Moving one layer off the last stage shifts a quarter of a stage's work and makes the imbalance *worse* (FIR $1.12 \rightarrow 1.23$). What is missing is a knob that changes a stage's cost in steps much smaller than a layer.

We find this knob in cross-layer KV sharing. Methods such as CLA (Brandon et al., 2024), MLKV (Zuhri et al., 2025), YOCO (Sun et al., 2024) and Echo-LLaMA (Dialameh et al., 2025) let a layer reuse the keys and values of an earlier layer. This saves KV-cache memory at inference

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.

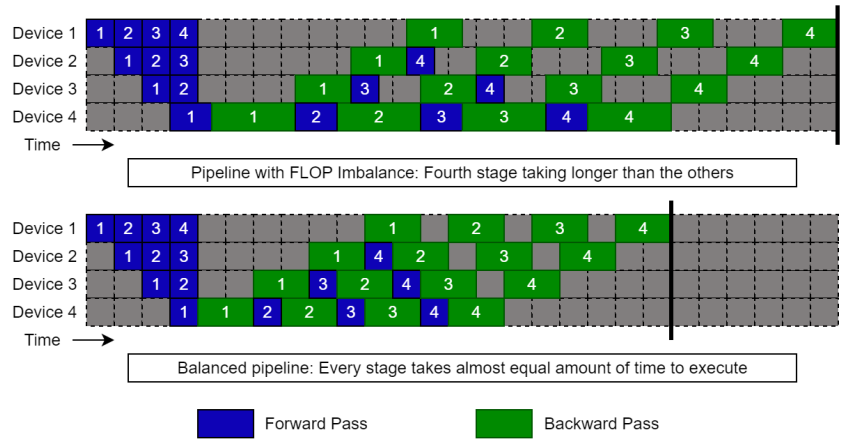


Figure 1: **Why stage balance matters.** Top: under 1F1B, a single heavier stage (here the last one, which also holds the LM head) stretches every steady-state step and leaves the other devices idle. Bottom: once the stages are balanced, the same schedule finishes earlier. KV-Pipe obtains this balance by converting a few attention layers on the heavy stage to cross-layer KV sharing.

and removes the layer's K/V projections. So far, these methods have been evaluated as memory or decoding optimizations, with sharing applied uniformly or to a fixed fraction of layers. In a pipeline, however, the same operation moves compute *between* stages: converting one LLaMA2-7B layer removes 0.14 layers' worth of work from exactly one stage. Four conversions on the last stage almost exactly cancel the LM-head surplus. This paper makes the following contributions:

- **A new use of KV sharing.** We treat the placement and amount of cross-layer KV sharing as a load-balancing knob for PP. It is orthogonal to the pipeline schedule and to partition search, and it does not change micro-batching or the optimizer (Section 3).
- **A simple, predictive cost model.** We define the FLOPs Imbalance Ratio (FIR). We show that when the imbalance comes from a per-stage surplus, the FIR-optimal number of shared layers depends only on the ratio of that surplus to the per-layer saving, not on the PP degree (Proposition 1). When we vary the LM-head surplus directly, the predicted budget matches the measured optimum in all five settings, and the measured speed-up is a stable 77–80% of the idealized one (Section 4.3).
- **The finding that balance, not FLOPs, determines PP efficiency.** Measured MFU is non-monotonic in the amount of sharing and follows FIR, not total FLOPs. At PP=8, sharing 2 or 16 layers gives the same FIR and the same MFU, although 16 layers remove $8\times$ more FLOPs (Section 4.2).
- **Evaluation on two hardware stacks.** On Ascend 910B NPUs, KV-Pipe improves MFU by up to 9.2% and reduces iteration time by up to 9.8%, with gains that grow with PP depth in a controlled sweep. It adds to Seq1F1B on GPUs for four model families, it keeps model quality within the variation of full-attention runs on LLaMA2-7B and LLaMA3-8B, and the same late-layer sharing speeds up GQA decoding by 7–8% (Section 4).

2 Stage Imbalance in Pipeline-Parallel LLMs

Consider a decoder-only Transformer with L layers partitioned into P pipeline stages, where stage i holds the layer set $\mathcal{S}_i$. Let F_ℓ be the FLOPs of layer ℓ for one micro-batch, and let C_i capture stage-specific components such as the embedding ($i{=}1$) or the LM head and loss ($i{=}P$). The work of stage i is $F_i = \sum_{\ell \in \mathcal{S}_i} F_\ell + C_i$.

Definition 1 (FLOPs Imbalance Ratio)**.**

$$\mathrm{FIR} = \frac{\max_i F_i}{\frac{1}{P}\sum_{i=1}^{P} F_i} \geq 1. \tag{1}$$

In the steady state of a 1F1B schedule, every step advances at the pace of the slowest stage. Ignoring warm-up, cool-down and communication, the fraction of device time lost to imbalance is therefore $1 - 1/\mathrm{FIR}$, and FIR$= 1$ means a perfectly balanced pipeline. FIR is computed from the partition and per-layer FLOP counts alone, which are available analytically or from one profiling pass. It is therefore a cheap, offline proxy for the bottleneck.

The LM-head surplus. With an even split ($|\mathcal{S}_i| = L/P$), the imbalance is driven by C_P. For LLaMA2-7B at S=8K and micro-batch size 1, the per-stage costs in Section B give $C_P/F_\ell = 0.556$: the LM head costs slightly more than half a decoder layer. At PP=8 this yields FIR$= 1.12$, and 10.7% of device time is idle in steady state. Because C_P is fixed while the per-stage work $A = \sum_{\ell \in \mathcal{S}_i} F_\ell$ shrinks as $1/P$, the imbalance grows with pipeline depth (FIR$= 1.018$, 1.051 and 1.12 at PP=2, 4 and 8, the first at S=4K). This is the regime in which deeper pipelines are used to fit larger models.

Why repartitioning cannot fix it. Partition search moves whole layers. At PP=8, one layer is 25% of a stage, which is almost twice the 13.9% surplus. Moving a layer from the last stage to its neighbour turns a $+13.9\%$ stage into a $+25\%$ stage (FIR $1.12 \to 1.23$). Uneven splits help only when the surplus is close to a whole number of layers, and the problem gets worse as P grows. A finer-grained way to change stage cost is needed.

3 KV-Pipe: KV Sharing as a Balancing Knob

Cross-layer KV sharing. A standard attention layer computes $\mathbf{Q}_\ell = \mathbf{X}_\ell \mathbf{W}_\ell^Q$, $\mathbf{K}_\ell = \mathbf{X}_\ell \mathbf{W}_\ell^K$, $\mathbf{V}_\ell = \mathbf{X}_\ell \mathbf{W}_\ell^V$. A KV-sharing layer instead reuses the keys and values of an earlier layer $\tau(\ell) < \ell$:

$$\mathbf{K}_\ell \leftarrow \mathbf{K}_{\tau(\ell)}, \qquad \mathbf{V}_\ell \leftarrow \mathbf{V}_{\tau(\ell)}. \tag{2}$$

The query projection, attention and MLP are unchanged. Converting layer ℓ removes its K/V projections, which is $4bsh^2/r$ forward FLOPs for batch b, sequence length s, hidden size h and GQA group ratio r (r=1 for MHA; derivation in Section J). It also removes one layer's entry from the KV cache during decoding. Let $z_\ell \in \{0, 1\}$ indicate a converted layer and write $F_\ell(z_\ell)$ with $\Delta F_\ell = F_\ell(0) - F_\ell(1) > 0$. Stage costs become $F_i(\mathbf{z}) = \sum_{\ell \in \mathcal{S}_i} F_\ell(z_\ell) + C_i$. For LLaMA2-7B at 8K, $\Delta F_\ell / F_\ell = 0.142$. A single conversion removes one quarter of the LM-head surplus.

Objective. Given a fixed partition, KV-Pipe chooses which layers to convert so as to balance the pipeline:

$$\min_{\mathbf{z} \in \{0,1\}^L} \mathrm{FIR}(\mathbf{z}) \qquad \text{s.t.} \qquad \textstyle\sum_\ell z_\ell \le m. \tag{3}$$

Unlike fixed-ratio sharing, the budget m is an upper bound and not a target. Sharing beyond balance lowers total FLOPs but also lowers the bottleneck stage below its peers, so FIR rises again. The following proposition makes this precise for the common case of a single heavy stage.

Proposition 1 (Balance-optimal budget)**.** *Suppose $P - 1$ stages have cost A and one stage has cost $A + H$ with $H > 0$. Suppose each conversion on the heavy stage removes Δ with $H/\Delta <$ the number of layers on that stage. Then $\mathrm{FIR}(m)$ is strictly decreasing for $m \le H/\Delta$ and strictly increasing for $m \ge H/\Delta$, so it is minimized at $m^\star \in \{\lfloor H/\Delta \rfloor, \lceil H/\Delta \rceil\}$. In particular, $m^\star$ does not depend on P, while the attainable reduction of the bottleneck stage, $H/(A+H)$, increases with P because $A \propto 1/P$.*

Proof sketch. For $m \le H/\Delta$ the heavy stage is the maximum and $\mathrm{FIR}(m) = \frac{A+H-m\Delta}{A+(H-m\Delta)/P}$, which is increasing in $H - m\Delta$ for $P > 1$. For $m \ge H/\Delta$ the maximum is A and $\mathrm{FIR}(m) = \frac{A}{A-(m\Delta-H)/P}$, which is increasing in m. The full proof is in Section D. □

For LLaMA2-7B at 8K, $H/\Delta = 0.556/0.142 = 3.9$, so $m^\star = 4$ at both PP=4 and PP=8 (FIR$= 1.0004$). The proposition also explains two regularities in our experiments: gains grow with PP, and oversharing hurts.

Placement algorithm. Algorithm 1 solves Equation (3) greedily. It repeatedly converts the deepest unconverted layer of the current bottleneck stage and recomputes FIR. It stops when FIR is within ϵ of 1, when the budget is spent, or when a further conversion would not reduce FIR. Starting from the

Algorithm 1 KV-Pipe: bottleneck-first KV-sharing placement

Require: partition $\{\mathcal{S}_i\}$; per-layer costs $F_\ell(0), F_\ell(1)$; stage costs C_i; tolerance ϵ; budget m; (optional) admissible layers $\mathcal{C}$

$\mathbf{z} \leftarrow \mathbf{0}$; $F_i \leftarrow \sum_{\ell \in \mathcal{S}_i} F_\ell(0) + C_i \ \ \forall i$
while FIR $> 1 + \epsilon$ **and** $\sum_\ell z_\ell < m$ **do**
  $i^\star \leftarrow \arg\max_i F_i$ ▷ ties broken toward the later stage
  $\ell^\star \leftarrow \max\{\ell \in \mathcal{S}_{i^\star} \cap \mathcal{C} : z_\ell = 0\}$ ▷ deepest unconverted layer
  if $\ell^\star$ is undefined **then break**
  end if
  $z_{\ell^\star} \leftarrow 1$; $F_{i^\star} \leftarrow F_{i^\star} - \Delta F_{\ell^\star}$
  if FIR did not decrease **then** undo the last conversion; **break**
  end if
end while
return $\mathbf{z}$, $m^\star = \sum_\ell z_\ell$

Table 1: **KV-Pipe on $8\times$ Ascend 910B (LLaMA2-7B, 1F1B).** Baseline is full attention; KV-Pipe uses the MFU-best SKV budget, with the FIR-optimal budget $m^\star$ from Algorithm 1 shown for reference. Relative changes are with respect to the baseline. FIR values are computed from the stage costs (Section B).

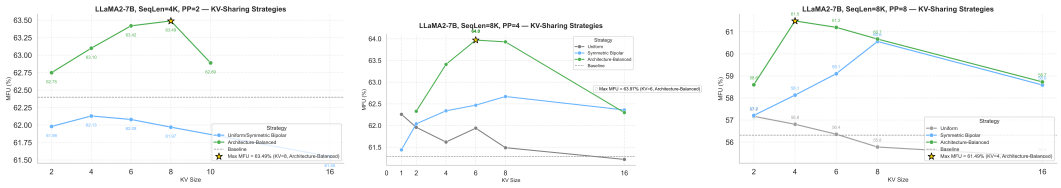
Setting	SKV (layers)	$m^\star$	MFU (%)	Iter. time (s)	FIR
S=4K, PP=2	8 (25–32)	5	$62.45 \rightarrow \mathbf{63.49}$ (+1.7%)	$104.14 \rightarrow \mathbf{99.27}$ (-4.7%)	$1.018 \rightarrow 1.010$
S=8K, PP=4	6 (27–32)	4	$61.28 \rightarrow \mathbf{63.97}$ (+4.4%)	$241.96 \rightarrow \mathbf{226.6}$ (-6.3%)	$1.051 \rightarrow 1.009$
S=8K, PP=8	4 (29–32)	4	$56.32 \rightarrow \mathbf{61.49}$ (+9.2%)	$263.30 \rightarrow \mathbf{237.5}$ (-9.8%)	$1.119 \rightarrow 1.0004$

bottleneck (usually the tail, because of the LM head) spends every conversion on the critical path. Choosing the deepest layer first keeps sharing in late layers, the regime used by prior late-layer KV sharing (Dialameh et al., 2025). If the bottleneck moves, for example when C_1 is large or the model is heterogeneous, the $\arg\max$ follows it. The procedure needs only the partition and per-layer FLOP counts, runs offline in $O(mP)$ time, and changes neither the schedule, the micro-batching nor the optimizer. Users who need a quality guarantee can restrict conversions to an admissible set $\mathcal{C}$, for example the second half of the network (Section F.1). In all our experiments the selected layers already lie in the last quarter of the model.

Budget refinement. FIR counts forward FLOPs. It does not see backward-pass asymmetries, loss computation, or communication. KV-Pipe therefore treats $m^\star$ from Algorithm 1 as the starting point of an optional short refinement. It measures a few budgets $m \in \{m^\star, \ldots, m^\star + 4\}$ that continue the same deepest-first order on the last targeted stage, and keeps the fastest. Each candidate needs only a few training iterations. Section 4.2 shows that without refinement, $m^\star$ is already the best budget at PP=8 and recovers 80% of the best measured MFU gain at PP=4.

4 Experiments

Setup. Our primary platform is $8\times$ Ascend 910B NPUs running MindSpeed-LLM with the standard non-interleaved 1F1B schedule. We train LLaMA2-7B (Touvron et al., 2023) in three configurations: (S=4K, PP=2), (S=8K, PP=4) and (S=8K, PP=8), with micro-batch size 1. To isolate pipeline effects, we use no tensor or context parallelism, and global batch size, precision and optimizer are fixed within each configuration (Section A). We sweep the number of shared-KV layers (SKV $\in \{2, 4, 6, 8, 16\}$, plus 1 and 10 where noted) under three placements (Section C): Uniform spreads conversions over stages, Symmetric-Bipolar splits them between a middle stage and the tail, and Bottleneck-first is KV-Pipe. We report model FLOPs utilization, $\mathrm{MFU} = F_{\text{model}} \cdot (\text{iterations/s}) / F_{\text{peak}}$, where F_{model} counts the FLOPs of the *modified* model, so work removed by sharing is not credited. We also report iteration time. Section 4.5 adds four models on $8\times$V100 GPUs, and Section 4.6 evaluates decoding.



(a) S=4K, PP=2 (b) S=8K, PP=4 (c) S=8K, PP=8

Figure 2: **MFU versus number of shared-KV layers** on 8× Ascend 910B for the three placements (dashed: full-attention baseline; star: best point in Table 1; Architecture-Balanced = KV-Pipe). Bottleneck-first placement dominates at every budget, and MFU peaks at a small, intermediate budget. At PP=2 the Uniform and Symmetric-Bipolar placements coincide. Iteration-time and FLOPs sweeps are in Section E.

Table 2: **Controlled effect of pipeline depth** (LLaMA2-7B, S=8K, P devices, no data parallelism, identical workload). The FIR-optimal budget is four layers at every depth, while the benefit grows as the fixed LM-head surplus becomes a larger fraction of each stage. "Predicted" is the idealized bottleneck reduction $H/(A+H)$ from Proposition 1. The PP=8 row is the configuration of Table 1.

PP	Baseline FIR	$m^\star$	FIR after	Baseline MFU (%)	KV-Pipe MFU (%)	ΔIter. time	Predicted
2	1.017	4	1.0004	62.8	63.6	−1.6%	−3.4%
4	1.051	4	1.0004	60.9	63.4	−5.4%	−6.5%
8	1.119	4	1.0004	56.3	61.5	−9.8%	−12.2%

4.1 Main Results: Gains Grow with Pipeline Depth

Table 1 and Figure 2 summarize the NPU results. KV-Pipe improves MFU and iteration time in every configuration, and the gain grows with pipeline depth: MFU improves by 1.7%, 4.4% and 9.2%, and iteration time falls by 4.7%, 6.3% and 9.8% at PP=2, 4 and 8. This is the trend predicted by Proposition 1. The LM-head surplus is a fixed cost, so it becomes a larger fraction of a stage as stages shrink. The idealized cost model, in which step time equals the bottleneck stage cost, predicts bottleneck reductions of 3.6%, 6.5% and 12.2%. The measured time reductions (4.7%, 6.3%, 9.8%) track these values. At PP=8 they are smaller, because warm-up, cool-down and communication do not benefit from balancing. At PP=2, where imbalance is small, part of the gain comes from the FLOPs that sharing removes (3.4% at SKV=8).

The PP=8 configuration isolates the balancing effect most clearly. The four shared layers remove only about 1.5% of model FLOPs (4473→4405 TFLOPs per iteration, Section E), yet iteration time falls by 9.8%. Almost all of the speed-up therefore comes from removing idle time, not from doing less work.

Controlled scaling with pipeline depth. The three configurations in Table 1 differ in more than PP: PP=2 uses S=4K, and PP=2 and 4 also use data parallelism on the remaining devices. To isolate pipeline depth, we repeat the experiment with everything else fixed: S=8K, the same micro-batch size, number of micro-batches, precision, optimizer and workload, and exactly P devices with no data parallelism. Since H and Δ do not depend on P, Proposition 1 predicts the same budget $m^\star = \lceil 3.9 \rceil = 4$ at every depth. Only the size of the surplus relative to a stage, and hence the headroom, should change.

Table 2 confirms both predictions. First, the same four bottleneck-first conversions bring FIR to 1.0004 at every depth, although the baseline FIR rises from 1.017 to 1.119. Second, the benefit grows monotonically with depth. MFU improves by 1.3%, 4.1% and 9.2%, and iteration time falls by 1.6%, 5.4% and 9.8%, which is 47–83% of the idealized bottleneck reduction. With sequence length, data-parallel degree and batch configuration held fixed, the larger gains of deeper pipelines can be attributed to stage imbalance.

Table 3: **FIR predicts MFU; FLOPs do not** (LLaMA2-7B, S=8K, PP=8). Top: three placements at the same budget. Bottom: bottleneck-first budget sweep. FIR is computed offline from stage costs; FLOPs are per iteration. Values marked ≈ are read from the sweep plots in Section E.

	SKV	Layers	FIR	FLOPs (TF)	Iter. time (s)	MFU (%)
UNIFORM	4	20, 24, 28, 32	1.104	4405	≈257.1	56.8
SYMMETRIC-BIPOLAR	4	15, 16, 31, 32	1.068	4405	≈251.2	58.1
BOTTLENECK-FIRST	4	29–32	**1.0004**	4405	**237.5**	**61.49**
Full attention	0	–	1.119	4473	263.3	56.32
BOTTLENECK-FIRST	2	31–32	1.059	4439	≈251.1	58.6
$= m^\star$	4	29–32	**1.0004**	4405	237.5	**61.49**
	6	27–32	1.009	4371	**236.8**	61.2
	8	25–32	1.019	4337	≈237.0	60.7
	16	17–32	1.057	4202	≈237.3	58.7

4.2 BALANCE, NOT FLOPS, DETERMINES PIPELINE EFFICIENCY

Table 3 tests the central claim directly. At a fixed budget of four layers, the three placements remove the same FLOPs. Yet their MFU differs by 4.7 points and follows FIR exactly: Uniform spreads the savings over stages that are not on the critical path (FIR 1.104, MFU 56.8%), while bottleneck-first removes the entire surplus (FIR 1.0004, MFU 61.49%).

The budget sweep shows the second half of the story. Total FLOPs fall monotonically with the number of shared layers, but MFU rises only until FIR reaches its minimum at $m^\star = 4$ and then declines. Past $m^\star$, further conversions push the former bottleneck *below* its peers and FIR rises again. This *over-correction* is visible in the iteration time, which plateaus at $\approx 237\,\mathrm{s}$ from 4 to 16 shared layers: the extra sharing buys no speed, because the critical path is now a different, unmodified stage. The most striking comparison is between 2 and 16 shared layers. They have almost identical FIR (1.059 vs. 1.057) and almost identical MFU (58.6% vs. 58.7%), even though 16 shared layers remove eight times more FLOPs. Across all eight PP=8 configurations in Table 3, FIR and MFU have Pearson correlation -0.995 and Spearman correlation -1.0.

FIR is a proxy, not an exact predictor. At PP=4, MFU peaks at SKV=6 rather than at $m^\star = 4$, and the correlation over the six sweep points is -0.80. The FIR-optimal budget alone still gives $+3.5\%$ MFU (63.41%), which is 80% of the best measured gain. At PP=8, $m^\star$ is itself the best budget. This motivates the short measured refinement in Section 3. Section G further compares FIR guidance with guidance from measured stage times, and it evaluates the admissible-layer guardrail.

4.3 PREDICTING THE BALANCE POINT BEFORE THE SWEEP

The sweeps above show that oversharing hurts once the pipeline is balanced. A stronger test of the cost model is whether it predicts *where* the balance point lies before any sweep is run. Proposition 1 states that $m^\star \in \{\lfloor H/\Delta \rfloor, \lceil H/\Delta \rceil\}$, so the optimal budget should move linearly with the last-stage surplus H. The experiments so far see only one natural value of H per model. We therefore vary H directly. In isolated performance runs (LLaMA2-7B, S=8K, PP=8), we change only the effective width of the LM-head projection, which changes the surplus on the last stage. The decoder, partition, micro-batching and all other computation stay fixed. We measure H in units of one decoder layer. For each setting we compute H/Δ offline, predict $m^\star$, and then find the measured optimum independently with a bottleneck-first budget sweep. This ablation validates the systems cost model only; the modified heads are not used for quality comparisons.

Table 4 shows that the prediction holds exactly. As the surplus grows from 0.28 to 0.85 layer equivalents, the predicted budget moves from two to six conversions, and in all five settings the measured optimum equals the prediction. The minimized FIR stays within 0.001 of one. The attainable time reduction grows with the surplus, from 5.1% to 13.6%, and is a nearly constant 77–80% of the idealized reduction $H/(A + H)$. The remaining 20% is consistent with the part of each iteration spent outside the balanced steady state, such as pipeline warm-up and cool-down, communication and the optimizer step. FIR is therefore not just an after-the-fact correlate of MFU.

Table 4: **LM-head surplus ablation** (LLaMA2-7B, S=8K, PP=8). The last-stage surplus H is varied while the decoder and partition stay fixed. The predicted budget $m^\star \approx H/\Delta$ is computed offline; the measured best m is the fastest point of an independent budget sweep. The row $H/F_\ell = 0.56$ is the unmodified model.

H/F_ℓ	H/Δ	Predicted $m^\star$	Measured best m	Baseline FIR	FIR at best m	ΔIter. time
0.28	2.0	2	2	1.062	1.001	-5.1%
0.43	3.0	3	3	1.092	1.001	-7.5%
0.56	3.9	4	4	1.119	1.0004	-9.8%
0.71	5.0	5	5	1.152	1.001	-11.7%
0.85	6.0	6	6	1.182	1.001	-13.6%

Table 5: **Quality and efficiency versus fixed-ratio sharing** (LLaMA2-7B, S=8K). Echo-style sharing converts a fixed 25% of layers (25–32) (Dialameh et al., 2025). Downstream Avg. averages MMLU, HellaSwag, ARC-Challenge and TruthfulQA. Changes are relative to the full-attention run of the same setting. PP=2 is in Section F.

Setting	Method (shared layers)	Val. PPL ↓	Downstream Avg. ↑	ΔMFU	ΔIter. time
PP=4	Full attention	5.67	52.5	–	–
	Echo-style 25% (8: 25–32)	5.50	52.0	+4.3%	-7.0%
	KV-Pipe (6: 27–32)	5.50	52.0	+4.4%	-6.3%
PP=8	Full attention	5.47	52.2	–	–
	Echo-style 25% (8: 25–32)	5.53	52.0	+7.8%	$\approx -10.0\%$
	KV-Pipe (4: 29–32)	5.49	52.2	**+9.2%**	-9.8%

From two numbers that are available before training, the head surplus and the per-conversion saving, it predicts the optimal budget and the attainable gain.

4.4 Model Quality and Comparison with Fixed-Ratio Sharing

KV sharing changes the architecture, so its efficiency must be weighed against quality. Table 5 reports validation perplexity and downstream accuracy for the configurations chosen by KV-Pipe, compared with full attention and with a fixed Echo-style 25% sharing ratio (protocol in Section F). All configurations selected by KV-Pipe convert only layers in the last quarter of the network. Their perplexity is never worse than that of the corresponding full-attention run by more than 0.02, and their downstream average is within 0.5 points. Both differences are smaller than the variation between the two full-attention runs at S=8K (5.47 vs.5.67 PPL). We therefore read these results as *no measurable quality loss* in the tested LLaMA2-7B settings, not as a quality improvement.

The comparison with fixed-ratio sharing isolates the value of choosing the budget from the pipeline. Converting 8 instead of 4 layers at PP=8 gives essentially the same iteration time ($\approx$ 237 s, the plateau in Table 3) but lower MFU, and it doubles the architectural change. KV-Pipe reaches the same speed with half the shared layers and the best quality in the group. At PP=4, the fixed ratio is 0.7 points faster in iteration time, using two more shared layers. A fixed ratio is right only by coincidence, whereas FIR identifies the budget beyond which sharing no longer helps the pipeline.

Quality on a GQA model. The LLaMA2-7B study uses multi-head attention. To check that small late-layer sharing is also benign for modern grouped-query models, we repeat the quality study on LLaMA3-8B with three runs per variant (Table 6). Sharing KV in the last four layers leaves perplexity within one standard deviation of full attention (6.23 ± 0.02 vs. 6.21 ± 0.02) and changes the downstream average by only 0.1 points. Each task moves by at most 0.1 points. The fixed 25% ratio, which changes twice as many layers, raises perplexity by 0.07, well outside the run-to-run spread, and lowers the downstream average by 0.3 points. The same pattern appears on LLaMA2-7B at PP=8: the quality cost of sharing grows with the number of converted layers, so sharing only what the pipeline needs is also the better choice for quality.

Table 6: **Quality on a modern GQA model** (LLaMA3-8B). The small tail-first configuration converts the same four late layers as KV-Pipe on LLaMA2-7B at PP=8; the fixed-ratio baseline converts the last 25% of layers. Validation PPL is the mean ± standard deviation over three runs; all variants share data, token budget, checkpoint protocol, optimizer, schedule and evaluation harness.

Method	Shared layers	Val. PPL ↓	MMLU	HellaSwag	ARC-C	TruthfulQA	Avg. ↑
Full attention	0	6.21 ± 0.02	65.1	79.2	53.8	44.2	60.6
Fixed 25%	8 (25–32)	6.28 ± 0.03	64.8	78.9	53.4	44.0	60.3
Tail-first, 4 layers	4 (29–32)	6.23 ± 0.02	65.0	79.1	53.7	44.2	60.5

Table 7: **Four models on $8\times$V100 GPUs: MFU (%)** at PP=2 / 4 / 8. The last column applies KV-Pipe on top of the Seq1F1B schedule. Iteration times are in Table 15.

Model	S	1F1B	Seq1F1B	Seq1F1B+KV-Pipe
LLaMA2-7B	4K	46.5/44.0/38.0	49.5/48.0/45.0	**51.5/50.0/48.0**
	8K	48.0/46.0/39.0	50.0/49.5/47.0	**52.0/52.0/50.0**
	16K	49.0/46.5/40.0	51.0/50.0/48.0	**53.0/52.5/51.0**
LLaMA2-13B	4K	47.0/44.5/38.5	49.0/48.5/46.0	**51.0/50.5/49.0**
	8K	48.5/45.0/39.0	50.0/49.5/47.5	**52.0/52.5/50.5**
	16K	49.5/45.5/39.5	51.0/50.0/48.5	**53.0/53.0/52.0**
LLaMA3-8B	4K	46.0/43.5/37.5	49.0/47.8/45.0	**51.0/49.8/48.0**
	8K	47.5/45.0/38.5	49.5/49.0/46.8	**51.5/51.8/49.8**
	16K	48.5/45.2/39.0	50.5/49.6/47.8	**52.5/52.0/51.0**
Qwen2.5-14B	4K	47.5/45.0/39.0	49.8/49.5/47.0	**52.0/51.8/50.0**
	8K	49.0/45.5/39.5	50.5/50.2/48.0	**52.5/53.0/51.0**
	16K	50.0/46.0/40.0	51.5/51.0/49.0	**53.5/54.0/52.5**

4.5 Generality across Models, Schedules and Hardware

To test whether the effect depends on the hardware, the model or the schedule, we repeat the evaluation on $8\times$ NVIDIA V100 GPUs with LLaMA2-7B/13B, LLaMA3-8B (Dubey et al., 2024) and Qwen2.5-14B (Qwen Team, 2024), for $S \in \{4, 8, 16\}$K and PP$\in \{2, 4, 8\}$ (Table 7). We compare 1F1B (Harlap et al., 2018), the stronger Seq1F1B schedule (Ao et al., 2025), and KV-Pipe applied on top of Seq1F1B. Seq1F1B reduces bubbles by splitting sequences, but it cannot remove the extra cost of the last stage. Adding KV-Pipe raises MFU over Seq1F1B by 3.9–4.4% at PP=2, 4.1–6.1% at PP=4 and 6.2–7.2% at PP=8, in every model and context length. For example, for LLaMA2-13B at 16K and PP=8, Seq1F1B+KV-Pipe reaches 52.0% MFU, compared with 48.5% for Seq1F1B and 39.5% for 1F1B. The same ordering and the same growth with PP hold on both hardware stacks. This supports the view that schedule-level bubble hiding and stage-level cost balancing address different sources of inefficiency. One caveat applies to the GQA models. For LLaMA3-8B and Qwen2.5-14B, a conversion removes only about 0.03 layers of forward FLOPs, and their large-vocabulary LM heads cost about two layers (Section J). The FLOP-based cost model therefore predicts that a few conversions can reduce, but not remove, their last-stage surplus. We report the gains for these models as empirical results and do not attribute them to FLOP balancing alone. Seq1F1B is not available in our Ascend software stack, which is why this composability study is run on GPUs (Section H). Section I further shows that KV-Pipe adds throughput on top of the DawnPiper partitioner (Peng et al., 2025).

4.6 The Same Shared Layers Also Speed Up Inference

Because KV-Pipe changes the architecture rather than the runtime, the shared layers stay after training. At inference they remove the corresponding KV-cache entries and K/V projections. Grouped-query attention (Ainslie et al., 2023) already shrinks K/V by the group ratio r, so the per-layer saving drops to $4h^2/r$ FLOPs and $2(h/r)$ cache elements per token (Section J). The saving is smaller, but it is not zero. Table 18 in Section K shows that sharing KV in the last 12.5% of layers improves long-context GQA decoding throughput by 7.3% (LLaMA3-8B, 32K) and 8.2% (Qwen2.5-14B,

128K). Decoding at these context lengths is dominated by reading the KV cache, and the shared layers shrink the cache by 12.5%. With multi-head attention and a larger sharing ratio, the effect is much larger. Section K reports $1.2\times$–$2.8\times$ decoding speed-ups for LLaMA2-7B with 50% sharing. KV-Pipe thus provides a second benefit: a model made pipeline-friendly for training is also cheaper to serve.

5 Related Work

Pipeline scheduling. GPipe (Huang et al., 2019) and PipeDream (Narayanan et al., 2019; Harlap et al., 2018) introduced micro-batched and 1F1B pipelines. Megatron-LM added interleaved stages (Shoeybi et al., 2019; Narayanan et al., 2021), and zero-bubble schedules (Qi et al., 2024), Seq1F1B (Ao et al., 2025) and SlimPipe (Li et al., 2025) further shrink bubbles. PipeFill (Arfeen et al., 2025) fills the remaining bubbles with other jobs. All of these methods *hide* idle time given fixed stage costs. KV-Pipe instead *removes* the stage-cost asymmetry that creates it, and it composes with them (Section 4.5).

Partitioning and automatic parallelism. DAPPLE (Fan et al., 2020), vPipe (Zhao et al., 2022), BPipe (Kim et al., 2023), DawnPiper (Peng et al., 2025), DynaPipe (Jiang et al., 2024b), Varuna (Athlur et al., 2022), Galvatron (Miao et al., 2022) and Unity (Unger et al., 2022) search over layer placements, memory, or joint parallelization strategies. They operate at the granularity of layers or operators, which is too coarse for the sub-layer LM-head surplus (Section 2). KV-Pipe changes the costs that such planners see, so it can serve as a pre-processing step for any of them.

Cross-layer KV sharing and KV-cache reduction. MQA (Shazeer, 2019) and GQA (Ainslie et al., 2023) share K/V across heads within a layer. CLA (Brandon et al., 2024), MLKV (Zuhri et al., 2025), YOCO (Sun et al., 2024) and Echo-LLaMA (Dialameh et al., 2025) share them across layers, and Wu et al. (2025b) compare such variants systematically. HShare (Wu et al., 2025a) shares critical KV indices, and PagedAttention (Kwon et al., 2023) manages the cache at serving time. Parameter sharing across depth goes back to ALBERT (Lan et al., 2019). These works decide how much to share in order to trade memory against quality, and they place sharing uniformly or by a fixed ratio. To our knowledge, KV-Pipe is the first to decide *where* and *how much* to share based on the pipeline's stage costs. It also shows that sharing beyond the balance point can reduce training throughput.

Heterogeneous models and serving pipelines. Sliding-window attention (Jiang et al., 2023), MoE layers (Jiang et al., 2024a) and Transformer–SSM hybrids (Lieber et al., 2024; Blakeman et al., 2025) make per-layer costs increasingly non-uniform. This widens the imbalance that KV-Pipe targets. Serving systems handle prefill–decode heterogeneity by scheduling (Agrawal et al., 2023; 2024; Guo et al., 2025; Zhang et al., 2025; Su et al., 2025), which is complementary to changing layer costs.

6 Conclusion

Pipeline-parallel LLM training loses a substantial fraction of its throughput to a structural, sub-layer imbalance that neither schedules nor layer partitioning can remove. We showed that cross-layer KV sharing, placed on the bottleneck stage and budgeted by a simple imbalance ratio, is a fine-grained knob that removes this imbalance. It improves MFU by up to 9.2% while removing only 1.5% of FLOPs, its benefit grows with pipeline depth under controlled conditions, it composes with stronger schedules, and it makes the trained model cheaper to serve. The underlying cost model is predictive: the ratio of the stage surplus to the per-conversion saving gave the measured optimal budget in every setting where we varied the surplus. More broadly, the results suggest that architectural choices usually made for inference efficiency, such as where to share KV, should also be made with the training system in mind.

References

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, 2024.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Sun Ao, Weilin Zhao, Xu Han, Cheng Yang, Xinrong Zhang, Zhiyuan Liu, Chuan Shi, and Maosong Sun. Seq1f1b: Efficient sequence-level pipeline parallelism for large language model training. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8998–9008, 2025.

Daiyaan Arfeen, Zhen Zhang, Xinwei Fu, Gregory Ganger, and Yida Wang. Pipefill: Using gpus during bubbles in pipeline-parallel llm training. *Proceedings of Machine Learning and Systems*, 7, 2025.

Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: Scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys)*, 2022.

Aaron Blakeman, Aarti Basant, Abhinav Khattar, Adithya Renduchintala, Akhiad Bercovich, Aleksander Ficek, Alexis Bjorlin, Ali Taghibakhshi, Amala Sanjay Deshmukh, Ameya Sunil Mahabaleshwarkar, et al. Nemotron-h: A family of accurate and efficient hybrid mamba-transformer models. *arXiv preprint arXiv:2504.03624*, 2025.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In *Advances in Neural Information Processing Systems*, 2024.

Maryam Dialameh, Rezaul Karim, Hossein Rajabzadeh, Omar Mohamed Awad, Boxing Chen, Hyock Ju Kwon, Walid Ahmed, and Yang Liu. Echo-llama: Efficient caching for high-performance llama training. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 2252–2269, 2025.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, Lansong Diao, Xiaoyong Liu, and Wei Lin. Dapple: A pipelined data parallel approach for training large models, 2020. URL `https://arxiv.org/abs/2007.01045`.

Tianyu Guo, Xianwei Zhang, Jiangsu Du, Zhiguang Chen, Nong Xiao, and Yutong Lu. gllm: Global balanced pipeline parallelism systems for distributed llms serving with token throttling. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1725–1741, 2025.

Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. PipeDream: Fast and efficient pipeline parallel DNN training. *arXiv preprint arXiv:1806.03377*, 2018.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril,

Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024a. URL https://arxiv.org/abs/2401.04088.

Chenyu Jiang, Zhen Jia, Shuai Zheng, Yida Wang, and Chuan Wu. Dynapipe: Optimizing multi-task training through dynamic pipelines. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, pp. 542–559. ACM, April 2024b. doi: 10.1145/3627703.3629585. URL http://dx.doi.org/10.1145/3627703.3629585.

Taebum Kim, Hyoungjoo Kim, Gyeong-In Yu, and Byung-Gon Chun. Bpipe: Memory-balanced pipeline parallelism for training large language models. In *International Conference on Machine Learning*, pp. 16639–16653. PMLR, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Zhouyang Li, Yuliang Liu, Wei Zhang, Tailing Yuan, Bin Chen, Chengru Song, and Di Zhang. Slimpipe: Memory-thrifty and efficient pipeline parallelism for long-context llm training, 2025. URL https://arxiv.org/abs/2504.14519.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model, 2024. URL https://arxiv.org/abs/2403.19887.

Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. Galvatron: Efficient transformer training over multiple GPUs using automatic parallelism. *Proceedings of the VLDB Endowment*, 16(3):470–479, 2022.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pp. 1–15, 2019.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–15, 2021.

Xuan Peng, Xuanhua Shi, Haolin Zhang, Yunfei Zhao, and Xuehai Qian. Dawnpiper: A memory-scablable pipeline parallel training framework. *arXiv preprint arXiv:2505.05856*, 2025.

Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble pipeline parallelism. In *International Conference on Learning Representations*, 2024.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

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.

Qidong Su, Wei Zhao, Xin Li, Muralidhar Andoorveedu, Chenhao Jiang, Zhanda Zhu, Kevin Song, Christina Giannoula, and Gennady Pekhimenko. Seesaw: High-throughput llm inference via model re-sharding. *arXiv preprint arXiv:2503.06433*, 2025.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. In *Advances in Neural Information Processing Systems*, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, et al. Unity: Accelerating DNN training through joint optimization of algebraic transformations and parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.

Huaijin Wu, Lianqiang Li, Hantao Huang, Tu Yi, Jihang Zhang, Minghui Yu, and Junchi Yan. Hshare: Fast llm decoding by hierarchical key-value sharing. In *The Thirteenth International Conference on Learning Representations*, 2025a.

You Wu, Haoyi Wu, and Kewei Tu. A systematic study of cross-layer kv sharing for efficient llm inference. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 396–403, 2025b.

Hongbin Zhang, Taosheng Wei, Zhenyi Zheng, Jiangsu Du, Zhiguang Chen, and Yutong Lu. Td-pipe: Temporally-disaggregated pipeline parallelism architecture for high-throughput llm inference. In *Proceedings of the 54th International Conference on Parallel Processing*, pp. 689–698, 2025.

Shixiong Zhao, Fanxin Li, Xusheng Chen, Xiuxian Guan, Jianyu Jiang, Dong Huang, Yuhao Qing, Sen Wang, Peng Wang, Gong Zhang, Cheng Li, Ping Luo, and Heming Cui. vPipe: A virtualized acceleration system for achieving efficient and scalable pipeline parallel DNN training. *IEEE Transactions on Parallel and Distributed Systems*, 33(3):489–506, 2022.

Zayd Muhammad Kawakibi Zuhri, Muhammad Farid Adilazuarda, Ayu Purwarianti, and Alham Fikri Aji. Mlkv: Multi-layer key-value heads for memory efficient transformer decoding. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5516–5525, 2025.

A Experimental Setup Details

NPU experiments. $8\times$ Ascend 910B NPUs in a single node, MindSpeed-LLM with the standard (non-interleaved) 1F1B schedule, LLaMA2-7B, micro-batch size 1, and no tensor or context parallelism, so the remaining devices are used for data parallelism.

GPU experiments. We conduct experiments using two GPU configurations: a single node equipped with $8\times$ NVIDIA V100 GPUs and a single node equipped with $8\times$ NVIDIA A100 40 GB GPUs. We evaluate both the 1F1B and Seq1F1B pipeline schedules.

MFU. $\mathrm{MFU} = F_{\text{model}}\, T_{\text{iter}}/F_{\text{peak}}$, where F_{model} is the model's FLOPs per iteration (forward and backward) for the *modified* architecture, T_{iter} the measured iterations per second, and F_{peak} the aggregate peak of all devices. Because F_{model} decreases with sharing, an MFU gain means the iteration got faster by more than the FLOPs removed.

B Stage-Level Cost Tables

Tables 8 to 11 give per-stage FLOPs, forward time and memory for LLaMA2-7B, PP=8, S=8K, micro-batch size 1, for the baseline and for the three placements with four shared layers. Stages are 0-indexed; layers are 1-indexed, so stage i holds layers $4i{+}1$ to $4i{+}4$.

Calibrating the cost model. Let a denote the cost of one stage of four decoder layers. The exact FIRs of the baseline (1.119454) and of the bottleneck-first placement (1.000404) determine the LM-head surplus $H = 0.13889\,a$ (i.e., 0.556 layers) and the per-conversion saving $\Delta = 0.03553\,a$ (i.e., 0.142 layers). With these two numbers the model reproduces the independently computed FIRs of the Uniform (1.103805) and Symmetric-Bipolar (1.068261) placements to six decimals. All FIR values in Tables 1 and 3 for S=8K are computed from this model. At S=4K (PP=2), the reported FIRs 1.0182 and 1.0101 (8 shared layers) give $H = 0.148\,a'$ and $\Delta = 0.0285\,a'$, where a' is the cost of four layers at 4K, so $H/\Delta = 5.2$ and $m^\star = 5$.

Table 8: **Baseline** for LLaMA2-7B, PP=8, S=8K, MBS=1. The decoder layers are evenly split, but the last stage is the bottleneck because it also holds the LM head.

Stage	0	1	2	3	4	5	6	7
Emb	1	0	0	0	0	0	0	0
Full attn	4	4	4	4	4	4	4	4
KV sharing	0	0	0	0	0	0	0	0
LM head	0	0	0	0	0	0	0	1
FLOPs	1.55E+13	1.55E+13	1.55E+13	1.55E+13	1.55E+13	1.55E+13	1.55E+13	1.763E+13
FWD time	59712.13	59237.7	59237.7	59237.7	59237.7	59237.7	59237.7	66796.00
Memory	16.8425	15.5625	15.5625	15.5625	15.5625	15.5625	15.5625	16.8425

Table 9: **Uniform placement** (4 shared layers). The budget is spread over stages 4–7; stage 7 remains the bottleneck (FIR$=1.104$).

Stage	0	1	2	3	4	5	6	7
Emb	1	0	0	0	0	0	0	0
Full attn	4	4	4	4	3	3	3	3
KV sharing	0	0	0	0	1	1	1	1
LM head	0	0	0	0	0	0	0	1
FLOPs	1.55E+13	1.55E+13	1.55E+13	1.55E+13	1.49E+13	1.49E+13	1.49E+13	1.708E+13
FWD time	59712.13	59237.7	59237.7	59237.7	57301.7	57301.7	57301.7	64860.00
Memory	16.8425	15.5625	15.5625	15.5625	15.5597	15.5597	15.5597	16.8396

Table 10: **Symmetric-Bipolar placement** (4 shared layers). The budget is split between stages 3 and 7; stage 7 remains the bottleneck (FIR= 1.068).

Stage	0	1	2	3	4	5	6	7
Emb	1	0	0	0	0	0	0	0
Full attn	4	4	4	2	4	4	4	2
KV sharing	0	0	0	2	0	0	0	2
LM head	0	0	0	0	0	0	0	1
FLOPs	1.55E+13	1.55E+13	1.55E+13	1.44E+13	1.55E+13	1.55E+13	1.55E+13	1.653E+13
FWD time	59712.13	59237.7	59237.7	55365.7	59237.7	59237.7	59237.7	62924.00
Memory	16.8425	15.5625	15.5625	15.5566	15.5625	15.5625	15.5625	16.8366

Table 11: **Bottleneck-first placement (KV-Pipe)** (4 shared layers). All four conversions remove the tail surplus. Stage FLOPs become nearly equal (FIR= 1.0004), and stage 0 becomes marginally critical in measured forward time because of the embedding.

Stage	0	1	2	3	4	5	6	7
Emb	1	0	0	0	0	0	0	0
Full attn	4	4	4	4	4	4	4	0
KV sharing	0	0	0	0	0	0	0	4
LM head	0	0	0	0	0	0	0	1
FLOPs	1.55E+13	1.55E+13	1.55E+13	1.55E+13	1.55E+13	1.55E+13	1.55E+13	1.543E+13
FWD time	59712.13	59237.7	59237.7	59237.7	59237.7	59237.7	59237.7	59052.00
Memory	16.8425	15.5625	15.5625	15.5625	15.5625	15.5625	15.5625	16.8308

C PLACEMENT STRATEGIES

Figure 3 illustrates the three placements. They differ only in *where* a given budget is spent. **Uniform** converts the last layer of each of the last m stages, so the savings are spread across stages. **Symmetric-Bipolar** splits the budget between two anchors, the tail and the middle stage, converting the deepest layers of each. It is a natural choice when imbalance is not concentrated at the tail. **Bottleneck-first** (KV-Pipe, labelled "Architecture-Balanced" in the plots) converts the deepest layers of the current bottleneck stage (Algorithm 1). In the sweeps, budgets larger than $m^\star$ continue the same deepest-first order, i.e., they convert the last m layers. This probes over-correction.

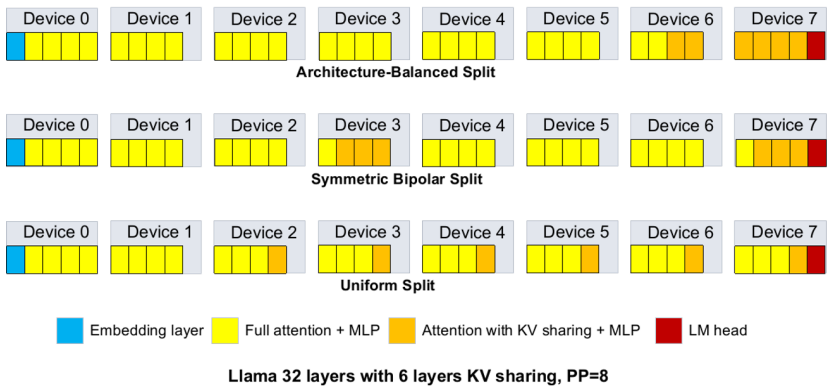


Figure 3: The three placements for a 32-layer LLaMA with six shared layers and PP=8.

D Proof of Proposition 1

Let the heavy stage have cost $A + H - m\Delta$ after m conversions and let the other $P - 1$ stages have cost A. The average is $\bar{F}(m) = A + (H - m\Delta)/P$.

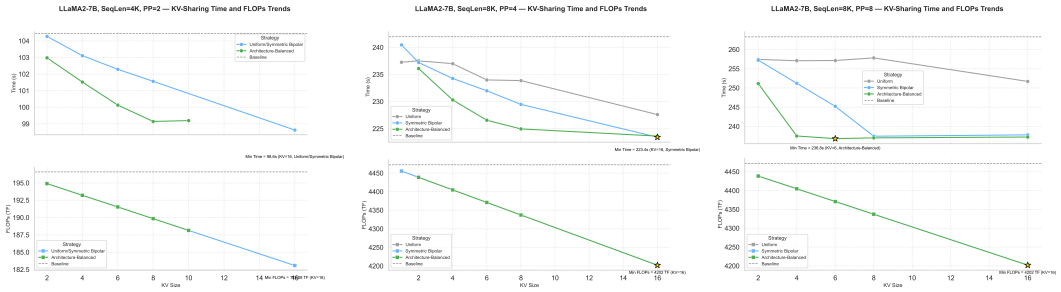
Case $m \leq H/\Delta$. The heavy stage is the maximum. Write $x = H - m\Delta \geq 0$. Then $\mathrm{FIR} = \frac{A+x}{A+x/P}$ and $\frac{d}{dx}\mathrm{FIR} = \frac{A(1-1/P)}{(A+x/P)^2} > 0$ for $P > 1$. Since x decreases in m, FIR is strictly decreasing in m.

Case $m \geq H/\Delta$. The heavy stage is now at most A, so the maximum is A and $\mathrm{FIR} = \frac{A}{A-(m\Delta-H)/P}$, which is strictly increasing in m as long as the denominator stays positive, which holds whenever the stage still has unconverted layers.

FIR is therefore unimodal with its minimum at $m = H/\Delta$, and over the integers at $\lfloor H/\Delta \rfloor$ or $\lceil H/\Delta \rceil$. Neither depends on P. In the idealized model, the step time equals the bottleneck cost, so at $m^\star$ it falls from $A + H$ to at most A, a relative reduction of $H/(A + H)$. For an even split, A is proportional to L/P, so this reduction increases with P. □

Predicted versus measured time reduction. With a the cost of four layers and $A = (L/P)\, a/4$: at S=8K, PP=8 has $A = a$ and $H/(A + H) = 12.2\%$, and PP=4 has $A = 2a$ and 6.5%. At S=4K, PP=2 has $A = 4a'$ and $H'/(A + H') = 3.6\%$. The measured reductions are 9.8%, 6.3% and 4.7% (Table 1).

E Full Iteration-Time and FLOPs Sweeps



(a) S=4K, PP=2 (b) S=8K, PP=4 (c) S=8K, PP=8

Figure 4: **Iteration time (top) and model FLOPs per iteration (bottom)** versus the number of shared-KV layers on 8× Ascend 910B. FLOPs fall linearly with sharing (about 17 TF per layer at 8K), whereas iteration time flattens once the bottleneck has been removed. The FLOPs of a given budget do not depend on placement, so the bottom panels show one line per budget.

Figure 4 shows the iteration time and FLOPs behind Figure 2. Two features support the balance interpretation of Section 4.2. First, bottleneck-first placement reaches its time plateau with the fewest shared layers (4 at PP=8). Symmetric-Bipolar needs 8 layers to approach it, and Uniform does not reach it even with 16. Second, the time-minimizing budget can be larger than the MFU-maximizing one. Past the balance point, time barely changes while FLOPs keep falling, so MFU declines.

F Model Quality

All downstream numbers average MMLU, HellaSwag, ARC-Challenge and TruthfulQA. In every tested configuration the earliest converted layer lies in the last quarter of the network's layers. This is an empirical property of these settings, where the LM-head stage is the bottleneck. It is not a guarantee for arbitrary models or partitions, which motivates the guardrail below.

Table 12: **Full quality–efficiency results for LLaMA2-7B.** Changes are relative to the full-attention run of the same setting. At PP=2, KV-Pipe and the 25% fixed ratio select the same layers (25–32), so they are the same model.

Setting	Method	SKV	Converted layers	Earliest	Val. PPL ↓	Downstream Avg. ↑	ΔMFU	ΔIter. time
S=4K, PP=2	Full attention	0	–	–	5.54	52.2	–	–
	KV-Pipe = Echo-style 25%	8	25–32	25	5.51	51.9	+1.7%	−4.7%
S=8K, PP=4	Full attention	0	–	–	5.67	52.5	–	–
	Echo-style 25%	8	25–32	25	5.50	52.0	+4.3%	−7.0%
	KV-Pipe	6	27–32	27	5.50	52.0	+4.4%	−6.3%
S=8K, PP=8	Full attention	0	–	–	5.47	52.2	–	–
	Echo-style 25%	8	25–32	25	5.53	52.0	+7.8%	≈ −10.0%
	KV-Pipe	4	29–32	29	5.49	52.2	+9.2%	−9.8%

F.1 Admissible-Layer Guardrail

For a minimum normalized depth $\alpha \in [0, 1)$, define $\mathcal{C}_\alpha = \{\ell : \ell \geq \lceil \alpha L \rceil\}$ and restrict the selection step of Algorithm 1 to $\ell^\star = \max\{\ell \in \mathcal{S}_{i^\star} \cap \mathcal{C}_\alpha : z_\ell = 0\}$. With $\alpha = 0.5$, sharing is confined to the second half of the network. If no admissible layer remains before FIR reaches $1 + \epsilon$, KV-Pipe stops and reports the residual imbalance rather than converting an earlier layer. The guardrail does not change any result in this paper, because every selected layer already satisfies $\alpha = 0.75$.

G Additional Ablations

Table 13: **Placement and guidance-signal ablations** (LLaMA2-7B, PP=8, S=8K, 4 shared layers). Efficiency rows for Uniform and Symmetric-Bipolar are taken from the sweeps in Figures 2 and 4.

Variant	Converted layers	FIR	Val. PPL ↓	Downstream Avg. ↑	ΔMFU	ΔIter. time
Uniform	20, 24, 28, 32	1.104	5.50	52.1	+0.9%	≈ −2.4%
Symmetric-Bipolar	15, 16, 31, 32	1.068	5.50	52.1	+3.2%	≈ −4.6%
Bottleneck-first, FIR-guided	29–32	1.0004	5.49	52.2	**+9.2%**	**−9.8%**
Bottleneck-first, stage-time-guided	28–31	–	5.49	52.2	+8.9%	−9.5%

Placement. At equal budget, only bottleneck-first placement removes the critical-path surplus. The other two placements spend part of the budget on stages that are already below the bottleneck, and their gains are correspondingly smaller. Quality is statistically indistinguishable across the three placements.

FIR versus measured stage times. Guiding the greedy selection with measured per-stage times instead of FIR gives a similar result (+8.9% vs. +9.2% MFU). FIR is intended as a cheap offline signal that needs no profiling. When accurate stage-time profiles are already available, they are an equally valid input to Algorithm 1.

Tolerance ϵ. The FIR trajectory in Table 3 determines the budget selected for any ϵ. Algorithm 1 stops at $m = 4$ for $\epsilon < 0.028$ and at $m = 3$ for $0.028 \leq \epsilon < 0.059$. It always stops at $m^\star = 4$ when run with $\epsilon = 0$ and the "FIR did not decrease" rule. We recommend $\epsilon = 0$ (or a small value) followed by the budget refinement of Section 3.

H Composability with Seq1F1B

Relative to Seq1F1B alone, adding KV-Pipe increases MFU by 6.4% (LLaMA2-7B) and 7.2% (LLaMA2-13B) and reduces iteration time by 6.4% and 8.5%. Seq1F1B requires schedule-level runtime support that exists in our CUDA stack but not in our Ascend 910B stack. The NPU experiments therefore use 1F1B and the composability study uses GPUs. This is a limitation of our software stacks, not a requirement of KV-Pipe, which only changes per-layer costs.

Table 14: **KV-Pipe on top of Seq1F1B** (8×V100, PP=8; subset of Table 7).

Model / setting	Schedule	MFU	Iter. time
LLaMA2-7B, S=8K	1F1B	39.0%	305 s
	Seq1F1B	47.0%	265 s
	Seq1F1B + KV-Pipe	**50.0%**	**248 s**
LLaMA2-13B, S=16K	1F1B	39.5%	960 s
	Seq1F1B	48.5%	820 s
	Seq1F1B + KV-Pipe	**52.0%**	**750 s**

Table 15: **Four models on 8×V100 GPUs: MFU and iteration time.** Each cell reports PP=2 / 4 / 8. The KV-Pipe columns apply KV-Pipe on top of the Seq1F1B schedule.

		MFU (%) at PP=2 / 4 / 8			Iteration time (s) at PP=2 / 4 / 8		
Model	S	1F1B	Seq1F1B	Seq1F1B+KV-Pipe	1F1B	Seq1F1B	Seq1F1B+KV-Pipe
LLaMA2-7B	4K	46.5/44.0/38.0	49.5/48.0/45.0	**51.5/50.0/48.0**	112/125/150	106/114/128	**101/109/120**
	8K	48.0/46.0/39.0	50.0/49.5/47.0	**52.0/52.0/50.0**	215/265/305	205/245/265	**196/232/248**
	16K	49.0/46.5/40.0	51.0/50.0/48.0	**53.0/52.5/51.0**	420/520/610	400/470/520	**385/448/485**
LLaMA2-13B	4K	47.0/44.5/38.5	49.0/48.5/46.0	**51.0/50.5/49.0**	195/220/260	186/204/228	**176/192/212**
	8K	48.5/45.0/39.0	50.0/49.5/47.5	**52.0/52.5/50.5**	365/430/505	350/398/445	**334/372/410**
	16K	49.5/45.5/39.5	51.0/50.0/48.5	**53.0/53.0/52.0**	705/820/960	675/740/820	**640/690/750**
LLaMA3-8B	4K	46.0/43.5/37.5	49.0/47.8/45.0	**51.0/49.8/48.0**	118/132/158	111/121/136	**105/114/126**
	8K	47.5/45.0/38.5	49.5/49.0/46.8	**51.5/51.8/49.8**	228/278/320	217/257/278	**205/240/258**
	16K	48.5/45.2/39.0	50.5/49.6/47.8	**52.5/52.0/51.0**	445/545/635	420/495/545	**402/468/505**
Qwen2.5-14B	4K	47.5/45.0/39.0	49.8/49.5/47.0	**52.0/51.8/50.0**	210/236/278	200/218/242	**188/202/222**
	8K	49.0/45.5/39.5	50.5/50.2/48.0	**52.5/53.0/51.0**	392/465/540	375/428/472	**356/398/430**
	16K	50.0/46.0/40.0	51.5/51.0/49.0	**53.5/54.0/52.5**	760/885/1030	725/800/880	**680/735/800**

I Comparison with Pipeline Partitioners

Table 16: **Comparison with pipeline partitioners** in the vPipe/DawnPiper setting (8× A100 40GB, 8 stages). Throughput is averaged over the feasible batch-size sweep; speed-up is relative to vPipe-AS. Stage imbalance is $\max_i t_i / \mathrm{avg}_i t_i$ from measured stage times.

Model	Method	Mode	Max batch	Samples/s	Speed-up	Imbalance
GPT-2 770M	GPipe (Huang et al., 2019)	Sync.	7	16.0	0.56×	1.06
	vPipe-S (Zhao et al., 2022)	Sync.	7	16.2	0.56×	1.05
	vPipe-AS	Async.	16	28.7	1.00×	1.05
	DawnPiper-AS (Peng et al., 2025)	Async.	20	33.0	1.15×	1.03
	KV-Pipe	Async. + SKV	20	**34.5**	**1.20**×	**1.02**
T5 780M	GPipe	Sync.	80	56.0	0.48×	1.42
	vPipe-S	Sync.	80	62.0	0.53×	1.31
	vPipe-AS	Async.	180	116.0	1.00×	1.25
	DawnPiper-AS	Async.	220	155.0	1.34×	1.12
	KV-Pipe	Async. + SKV	220	**163.0**	**1.41**×	**1.07**

KV-Pipe improves throughput over DawnPiper-AS by 4.5% (GPT-2) and 5.2% (T5), and it lowers the measured stage imbalance from 1.03 to 1.02 and from 1.12 to 1.07. We read these results as evidence that changing per-layer costs gives headroom beyond partition search, not that KV-Pipe replaces partitioners.

J Attention FLOPs under MHA and GQA

We count a multiply–accumulate as two FLOPs, so a product of $\mathbb{R}^{m\times k}$ and $\mathbb{R}^{k\times n}$ matrices costs $2mkn$. Let the batch size be b, sequence length s, hidden size h, number of query heads a, head

Table 17: **Positioning of KV-Pipe** relative to representative PP optimizations, by primary lever.

Method	Primary lever	Relation to KV-Pipe
GPipe, PipeDream, 1F1B	Micro-batched / asynchronous scheduling	Schedules fixed stage costs; KV-Pipe changes them.
Interleaved 1F1B, Zero-Bubble, Seq1F1B, SlimPipe	Finer-grained scheduling to shrink bubbles	Composable (Table 7).
PipeFill	Fills bubbles with other jobs	Uses idle time; KV-Pipe removes its cause.
DAPPLE, vPipe, DawnPiper, Varuna	Partition / placement search	Layer-granular; KV-Pipe provides sub-layer adjustment and can precede the search.
BPipe	Memory balancing across stages	Targets memory; KV-Pipe targets compute (and also frees KV memory).
DynaPipe	Dynamic micro-batching	Runtime adaptation vs. offline cost reshaping.
Galvatron, Unity	Joint parallelization search	Can search over the modified cost profile.

dimension $d = h/a$, number of KV heads g and $r = a/g \geq 1$ (the GQA group ratio), so the K/V width is h/r.

Projections. MHA computes three $h \times h$ projections: $\mathrm{FLOPs}_{\mathrm{QKV}}^{\mathrm{MHA}} = 6bsh^2$. GQA keeps **Q** full-size and shrinks K/V: $\mathrm{FLOPs}_{\mathrm{QKV}}^{\mathrm{GQA}} = 2bsh^2 + 2 \cdot 2bsh\,(h/r) = 2bsh^2(1 + 2/r)$.

Scores, weighted sum and output. $\mathbf{QK}^\top$ and $\mathrm{Attn} \cdot \mathbf{V}$ each cost $2bs^2h$ (per query head, so also under GQA), and the output projection costs $2bsh^2$.

Totals per token.

$$\mathrm{FLOPs/token}_{\mathrm{attn}}^{\mathrm{MHA}} = 8h^2 + 4sh, \qquad \mathrm{FLOPs/token}_{\mathrm{attn}}^{\mathrm{GQA}} = 4h^2 + \frac{4h^2}{r} + 4sh. \tag{4}$$

What sharing removes. A shared layer skips its K/V projections, saving $\Delta F = 4h^2/r$ forward FLOPs per token ($4h^2$ for MHA). It also skips $\Delta B = 2(h/r)\, b_{\mathrm{elem}}$ bytes of KV cache per token, where b_{elem} is the bytes per element. GQA reduces both savings by the factor r, but neither vanishes. For LLaMA2-7B (MHA, h=4096, MLP width 11008) at s=8K, the analytic forward saving is $4h^2$ out of $8h^2 + 4sh + 6h \cdot 11008$ per layer, i.e., 12.5%. This is consistent with the measured 14.2% (Section B), which also includes backward-related effects.

GQA models with large vocabularies. The same accounting at s=8K gives a very different picture for LLaMA3-8B (h=4096, r=4, MLP width 14336, vocabulary 128,256) and Qwen2.5-14B (h=5120, r=5, MLP width 13824, vocabulary 152,064). One conversion saves only 0.029 layers of forward FLOPs in both models, while the LM head ($2h \cdot |\mathcal{V}|$ per token) costs 1.84 and 2.17 layers, respectively. Hence $H/\Delta \approx 63$ and 74, far more conversions than a single stage holds. For such models, KV sharing can only partly offset the last-stage surplus. Complete balancing would have to combine it with moving layers off the last stage, which the coarse layer granularity allows here because the surplus is about two whole layers.

K Inference Results

With multi-head attention every layer stores full-width K/V, so cross-layer sharing has the most to remove. Figure 5 shows that sharing KV in half of the layers of LLaMA2-7B raises decoding throughput by $1.22\times$ at 8K and by 2.57–$2.77\times$ at 32K–128K. At long contexts the KV cache dominates memory and bandwidth, so this is where the gains are largest.

Table 18: **GQA decoding** with tail-first shared-KV layers covering 12.5% of the layers in both models.

Model, context	Variant	Throughput (tok/s)	Time/token (ms)	Speed-up
LLaMA3-8B, 32K	GQA baseline	620	1.61	–
	KV-Pipe, SKV=4	**665**	**1.50**	+7.3%
Qwen2.5-14B, 128K	GQA baseline	170	5.88	–
	KV-Pipe, SKV=6	**184**	**5.43**	+8.2%

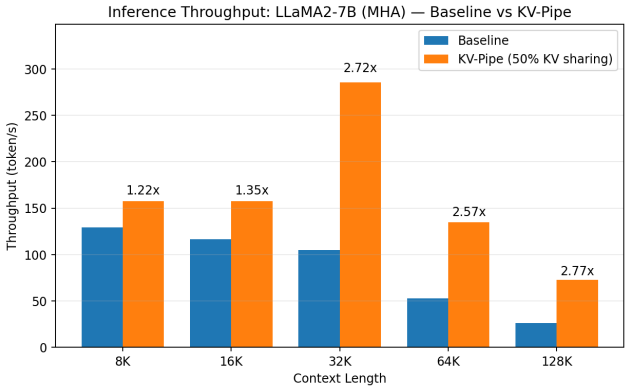


Figure 5: **Decoding throughput for LLaMA2-7B (MHA)** with 50% of layers sharing KV, compared with the full-attention baseline. This sharing ratio is larger than the budgets KV-Pipe selects for training. It illustrates the upper range of the inference-side benefit.

L Practical Guidance

KV-Pipe pays off most when (i) the PP degree is moderate to high, (ii) the partition contains non-uniform components such as a heavy LM head, a large vocabulary, or heterogeneous layers, and (iii) contexts are long enough that attention is a significant part of layer cost. A quick test is to compute FIR for the intended partition. If $1 - 1/\mathrm{FIR}$ is only a few percent (as at PP=2 in our experiments), the headroom is small. At PP=8 it was 10.7%, and KV-Pipe recovered most of it. In practice: compute $m^\star$ with Algorithm 1 ($\epsilon = 0$), optionally measure a few budgets above $m^\star$, and restrict conversions with the guardrail if quality guarantees are needed. Avoid sharing beyond the balance point. It saves FLOPs but not time, and it changes the architecture more.

Limitations

Although we evaluate KV-Pipe across multiple model families, context lengths, pipeline depths, and both 1F1B and Seq1F1B schedules, several limitations remain. First, our quality evaluation covers LLaMA2-7B and LLaMA3-8B; the remaining models provide efficiency evidence only. We therefore do not claim that cross-layer KV sharing is quality-neutral for substantially different architectures such as MoE, Transformer–SSM hybrids, or much deeper models. The admissible-layer guardrail (Section F.1) is intended to limit architectural changes when such quality evidence is unavailable.

Second, the effectiveness of KV sharing depends on the ratio between the stage-specific surplus H and the compute saving Δ of one conversion. For GQA models, Δ can be substantially smaller than for MHA (Section J); consequently, a large LM head may require more admissible shared layers than are available on the bottleneck stage. In this regime, KV-Pipe can reduce but may not fully eliminate the imbalance and should be combined with coarse layer repartitioning or other parallelism strategies.

Third, our experiments consider pipeline degrees up to PP$= 8$ on single-node configurations. The controlled PP sweep confirms that the benefit grows with pipeline depth under otherwise fixed con-

ditions, but at PP$=16$–32, inter-node communication and synchronization may become sufficiently large to change the critical stage or reduce the attainable gain from compute balancing.

Fourth, both the natural workloads and the controlled LM-head-surplus ablation place the initial bottleneck on the final stage. The surplus ablation shows that the predicted optimal sharing budget tracks changes in bottleneck severity, but it does not exercise the retargeting branch of Algorithm 1 in which the bottleneck begins on, or moves to, a different stage. Heterogeneous architectures with heavy intermediate or first-stage components would provide a stronger test of this behavior.

Finally, FIR is intentionally a lightweight forward-FLOP proxy rather than a complete runtime model. It does not explicitly capture communication, backward-pass asymmetries, kernel efficiency, or synchronization overhead. It identifies the optimal budget exactly at PP$=8$ and across the controlled surplus ablation, and remains close to the measured optimum in the shallower pipeline settings. For this reason, we use FIR to obtain a cheap initial configuration and retain the short measured refinement when runtime effects shift the optimum.

Reproducibility Statement

The cost model, FIR and the placement algorithm are fully specified in Sections 2 and 3 and Algorithm 1. The per-stage FLOP, time and memory tables that calibrate the model are in Section B, and the proof of Proposition 1 is in Section D. Hardware, software stacks, schedules and training settings are listed in Section A. The full sweeps behind Tables 1 and 3 are in Section E, the attention FLOP derivations in Section J, and the quality-evaluation protocol in Section F.

Ethics Statement

This work improves the computational efficiency of training and serving LLMs. Higher utilization lowers the energy and hardware cost of a given training run. It does not introduce new data, human subjects or capabilities beyond those of the underlying models, and we are not aware of ethical risks specific to it beyond those generally associated with LLM development.

AI Use Statement

We used generative AI tools to help edit the manuscript text, format the LaTeX source, and cross-check that numbers quoted in the text agree with the tables and figures. We did not use generative AI tools for research ideation, experiment design, implementation, running experiments, or producing experimental results. All AI-assisted text and consistency checks were reviewed and verified by the authors, who take full responsibility for the content of this paper.