DBO++: RETHINKING COMMUNICATION OVERLAP FOR DENSE TENSOR-PARALLEL PREFILL

Anonymous authors
Paper under double-blind review

ABSTRACT

Communication overlap can reduce prefill latency in dense language models, but its value depends on more than the input batch. We present DBO++, which extends vLLM's dual-batch execution to dense tensor-parallel prefill, and study how engine configuration changes the choice between overlapping and unsplit execution. On Qwen3-32B with four H200 GPUs, changing the configured batch-token limit while keeping the input and split fixed changes the mean effect from a 10.1% slowdown to a 3.6% latency reduction. Controlled comparisons connect this reversal to startup profiling and memory allocation. In the measured setting, live allocation increases allocator retries and imposes a penalty concentrated in the overlapping path. A targeted intervention preserves existing KV capacity while changing subsequent allocation handling; measured retries disappear and a 24.6% live-allocation penalty falls to approximately zero. Separately, a matched comparison with fixed KV allocation establishes a 2.1% latency reduction over unsplit execution. A rule using workload information and the configured limit preserves a token-only rule's positive savings while avoiding its two harmful choices in an initial held-out evaluation. Independent follow-up tests expose output mismatches and slower selections, delimiting this empirical rule's support. A broader four-model, one-output-token survey records both gains and regressions. Together, these results establish configuration-dependent overlap in the studied system and show why execution selection must be evaluated with the engine conditions and output contract under which it is used.

1 INTRODUCTION

Communication between GPUs can delay prompt processing in dense language models that use tensor parallelism. Computation is divided across GPUs, but collective operations such as all-reduce must combine partial results before dependent computation can continue (Narayanan et al., 2021). During this prompt-processing stage, known as prefill, splitting a batch allows one part to compute while the other communicates. The overlap can reduce waiting, but splitting and coordination add overhead. The system must therefore decide when overlap is faster than unsplit execution. TokenWeave, for example, uses an offline-profiled token threshold to decide when to split and overlap (Gond et al., 2026). We ask whether workload information alone remains sufficient when the model and hardware stay fixed but the engine configuration changes.

The same workload can favor different execution paths under different engine configurations. On Qwen3-32B with four H200 GPUs, we change only the configured batch-token limit while holding the input batch and its split fixed. This limit specifies the maximum number of tokens allowed in a batch, not the token count of the measured batch. The mean effect of overlap shifts from a 10.1% slowdown to a 3.6% latency reduction relative to unsplit execution. A fixed rule based only on the workload would make the same choice in both configurations, even though the faster path changes.

In the engine we study, the configured token limit also affects memory allocation during startup. We test the role of that allocation by varying the amount of memory held by the engine while keeping the workload fixed. This intervention changes the latency of overlapping execution but has little effect on unsplit execution or serial execution of the same split. These controls identify memory allocation as one contributor to the reversal. The finding motivates using engine configuration alongside workload information when deciding whether to overlap.

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.

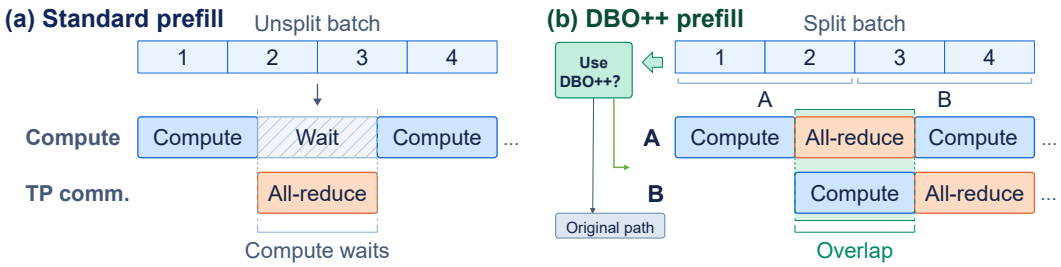

Figure 1: Dense tensor-parallel prefill on the same batch. (a) Standard execution waits for all-reduce before dependent computation resumes. (b) DBO++ splits the batch so that one microbatch computes while the other communicates. The admission rule retains the original path when overlap is not selected. Timings are schematic.

We present **DBO++**, which extends vLLM's dual-batch execution to dense tensor-parallel prefill (vLLM Project, 2026). DBO++ coordinates the two parts of a batch at all-reduce boundaries so that one part can compute while the other communicates (Figure 1). To decide when to use this path, a selector considers both workload information and the configured batch-token limit. The decision is made before splitting: if overlap is not selected, the batch remains on the original unsplit path. The selector uses metadata already available to the engine and requires no online GPU profiling.

In a separate matched Qwen3-32B experiment with cache allocation held fixed, the overlapping path reduces mean latency by 2.1% relative to unsplit execution. To evaluate the selector, we compare DBO++ with a token-only rule on 16 held-out workload–configuration cases, with both rules choosing between the same two execution paths. DBO++ preserves the token-only rule's positive latency savings while avoiding both cases in which that rule selects overlap even though unsplit execution is faster. Outputs from the paths selected by DBO++ match the unsplit reference exactly in this evaluation. Follow-up tests, however, reveal output differences and cases in which the selector chooses the slower path. The original results therefore do not establish output preservation or reliable selection beyond that evaluation. More broadly, a configured batch-token limit can affect not only how much work an engine allows, but also which execution path is faster for an unchanged batch.

Our contributions are:

1. **Dense prefill overlap.** We extend vLLM's dual-batch execution to dense tensor-parallel prefill and demonstrate a net latency reduction in a matched experiment with cache allocation held fixed.
2. **Configuration-dependent overlap.** We show that changing the configured batch-token limit can reverse which execution path is faster for a fixed batch and split. Controlled allocation experiments identify memory held by the engine as one contributor to this reversal.
3. **Configuration-informed selection.** We develop a pre-split selector that uses workload information and the configured batch-token limit. In the held-out evaluation, it preserves a token-only rule's positive latency savings while avoiding that rule's slower overlap choices.

2 Related Work

Overlap mechanisms. Communication can overlap with computation within GPU operators or across independently scheduled work. Kernel and compiler approaches fuse or coordinate computation and communication (Jangda et al., 2022; Chang et al., 2024; Zheng et al., 2025; Qiang et al., 2026). Execution-level systems interleave requests, partition sequences, or pipeline smaller batches (Du et al., 2024; Xiao & Su, 2024; Zhu et al., 2025). Centauri partitions communication for training schedules (Chen et al., 2024); T3 adds hardware support for fine-grained overlap (Pati et al., 2024). Ladder-Residual changes model dependencies to enable overlap (Zhang et al., 2025). DBO++ extends vLLM's existing dual-batch machinery, whose documented path coordinates microbatches around MoE communication (vLLM Project, 2026). It adds handoffs at dense all-reduce boundaries while reusing the engine's collectives and retaining the batch chosen by the scheduler.

Deciding when to overlap. Selective activation and resource-aware scheduling address the cost of overlap. TokenWeave combines a two-way token split with a fused AllReduce–RMSNorm kernel and an offline-profiled token threshold; smaller batches retain fusion without batch splitting (Gond et al., 2026). FlashOverlap uses models of computation, communication, and resource contention to tune overlap configurations (Hong et al., 2026). DynaFlow provides programmable operator schedules that can adapt to runtime batch conditions, with memory management for concurrent execution (Pan et al., 2026). These approaches establish that useful overlap requires both an execution mechanism and a choice of when and how to apply it. Our focus is whether changing the engine configuration can alter the preferred path even when the model, hardware, input batch, and split remain fixed. In the studied engine, changing only the configured batch-token limit can reverse which path is faster. DBO++ therefore includes that limit alongside workload information in its decision before splitting. The allocation experiments examine why this configuration can matter even when the current workload does not change.

Memory and serving configuration. Memory allocation and scheduling determine the conditions under which a batch executes. FlashAttention reduces memory traffic through tiling (Dao et al., 2022). PagedAttention reduces fragmentation and enables KV-cache sharing, while vAttention combines contiguous virtual addresses with on-demand physical allocation (Kwon et al., 2023; Prabhu et al., 2025). SGLang reuses shared prefixes with RadixAttention (Zheng et al., 2024). Scheduling changes which work runs together: Orca schedules at iteration boundaries (Yu et al., 2022); Sarathi-Serve and DeepSpeed-FastGen combine prefill chunks with decoding (Agrawal et al., 2024; Holmes et al., 2024), whereas DistServe and Splitwise place the two phases on separate resources (Zhong et al., 2024; Patel et al., 2024). DBO++ chooses how to execute an already scheduled batch, retaining its membership and KV capacity. Our allocation controls show that memory provisioning can change the preferred execution path for the same input.

3 DBO++

DBO++ adds an overlapping execution path for dense prefill without changing the batch selected by vLLM. Before splitting, a selector uses the batch shape and configured token limit to choose this path or the original unsplit path (Appendix Figure 5).

3.1 Executing Dense Prefill with Overlap

DBO++ divides the padded token range near its midpoint into two contiguous microbatches, A and B, with similar token counts. Both use the same dense model and TP group. Request membership and token positions remain unchanged. If a request crosses the split, its two portions remain one sequence: the runner adjusts query ranges and sequence lengths while retaining the original KV-cache mappings.

On each TP rank, A and B have separate CPU workers but share a compute stream and a communication stream. Queuing B's computation behind A's wait for all-reduce on the compute stream would prevent it from running during that wait. A therefore yields CPU control at all-reduce, letting B queue computation up to its next collective. B returns control, and A queues all-reduce on the communication stream. B's computation can then overlap with A's communication. The exchange repeats at later boundaries, using existing vLLM operators. CPU submission is sequential, while GPU execution can overlap.

The shared compute stream preserves submission order. CUDA events make each collective wait for its inputs and dependent computation wait for its result. Tensor storage remains valid until all streams finish using it. Once both microbatches finish, their outputs rejoin in input order. Output agreement is evaluated separately in the experiments. Compilation and failure handling are detailed in Appendix A.

3.2 Accounting for the Cost of Overlap

Overlap is useful only if the time saved exceeds the cost of splitting and coordination. The offline comparisons use an unsplit batch U, the same split run serially S, and the split run with DBO++ O. Comparing S with U measures the effect of splitting, including changes to operator shapes.

Comparing O with S measures the additional effect of enabling overlap, including its coordination cost. O versus U gives the net latency benefit. Any of these comparisons can favor either path.

Configuration can change this tradeoff even for a fixed workload. Let n be the batch's scheduled token count and B the configured maximum batch-token count. In the studied engine, B also affects startup memory allocation. A workload-only rule would make the same choice despite the observed reversal in path performance at fixed n and split. We therefore include B in the selector.

3.3 Selecting the Execution Path

The evaluated rule was calibrated for Qwen3-32B BF16 with TP4 on H200 and frozen before held-out evaluation. The selector requires complete metadata and a supported pure prefill batch with equal scheduled token counts per request. Such a batch is admitted only if it contains 10,240–20,480 scheduled tokens, inclusive, and this count is at most 75% of the configured batch-token limit. Otherwise, execution stays on the unsplit path.

These values come from development measurements. The token interval spans the smallest and largest workloads with observed benefit; the 75% boundary separates the observed beneficial and harmful n/B values within that interval (Appendix A). The first condition restricts workload size, while the second allows a fixed batch to receive different decisions across configurations.

The runtime decision uses existing metadata without online GPU profiling or cache resizing. The thresholds are empirical: later allocation studies explain why configuration can matter but did not determine these values. Section 4.4 reports both the original held-out results and the independent failures, which limit the rule's applicability.

4 Evaluation

The experiments reveal three consistent patterns. First, engine configuration can reverse the benefit of overlap for an unchanged input; the reversal is concentrated in the overlapping path. Second, live memory allocations affect that path selectively, and changing allocation handling removes the associated penalty. Third, configuration improves execution choices in the calibrated regime, while the broader survey finds the most consistent gains on long inputs. We establish each pattern with controlled comparisons, then examine its range across workloads and models.

4.1 Experimental Setup

The controlled studies use Qwen3-32B BF16 with TP4 on H200 (Qwen Team, 2025). Four input shapes each contain 20,480 scheduled tokens: $5 \times 4{,}096$, $10 \times 2{,}048$, $4 \times 5{,}120$ and $8 \times 2{,}560$, where the two dimensions are batch size and prompt length. Comparisons use fresh process blocks and hold the input and split fixed. The path and allocation controls use five warmups followed by 50 measurements per workload. The allocator experiments repeat measurements on two TP4 groups on the same node. Their confidence intervals account for workload and process variation, rather than treating individual timed calls as independent runs.

The broader survey measures four BF16 models—Qwen3-32B, Qwen3.5-27B, Llama-3.1-70B (Llama Team, 2024) and Gemma-3-27B (Gemma Team, 2025)—at TP2 and TP4, with Qwen3-32B TP1 as a control. Each eager invocation returns one output token. Its latency covers prompt processing, host submission and first-token sampling; subsequent decoding is outside this measurement. Controlled experiments instead time a synchronized forward region. Online serving results are reported separately in the supplement.

For candidate A and reference R, latency reduction is $100(1 - T_A/T_R)$, so a positive value indicates a faster candidate. Allocation penalties compare live with absent allocation within the same path, with positive values indicating added latency. Controlled results use the frozen 95% confidence intervals. Repeated survey measurements report the mean and sample standard deviation of process medians. Supplementary Table 3 specifies the measurement unit and output checks for each study.

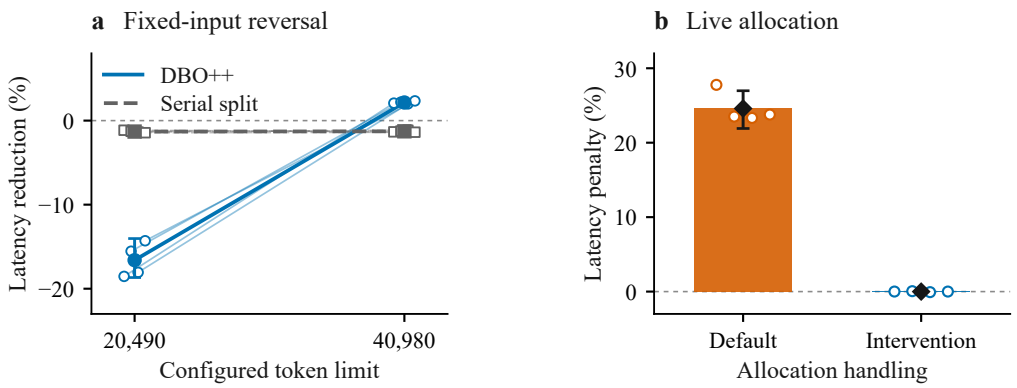

Figure 2: Configuration and allocation state change overlap performance. (a) Identical inputs at two token limits: DBO++ reverses its effect while serial splitting remains stable. Both use unsplit execution as reference. (b) In a separate intervention, changing allocation handling removes the live-allocation penalty within DBO++. Open points show four workload means; filled points and bars show aggregates and frozen 95% intervals. There are three process repetitions in (a), and three on each of two same-node TP4 groups in (b).

4.2 Configuration Reverses the Benefit of Overlap

The configured token limit changes which path is faster even when the input and split do not change. In the original factorial experiment, raising the limit from 20,490 to 40,980 changes DBO++ from a 10.10% slowdown to a 3.61% latency reduction. The capacity effect is 13.711 percentage points, with a 95% interval of $[13.153, 14.268]$. The registered model-length, sequence-count and interaction effects have intervals crossing zero. Thus the reversal follows a configuration change, rather than an increase in the work performed by the measured batch.

An independent three-path experiment shows where this difference arises (Figure 2a). Relative to unsplit execution, serial splitting costs about 1.3% at both limits. Full overlap changes from a 16.61% slowdown to a 2.15% reduction. With the split held fixed, the incremental effect of overlap changes from a 15.11% slowdown to a 3.39% reduction, whereas unsplit latency changes by only 0.064%. Across all four shapes, the overlap improvement is 16.35–20.57 points. The aggregate change is 18.492 points (95% interval $[16.029, 20.366]$), compared with 0.031 points for serial splitting. The stable serial control localizes the reversal to the conditions under which overlap executes.

4.3 Live Allocation Selectively Penalizes Overlap

Memory held by the engine contributes to the reversal through a penalty concentrated in overlapping execution. In this engine, a larger configured token limit increases the startup profiling workload and leaves a smaller final KV allocation. Crossing profile and allocation independently separates their effects: reducing the cache from 95,042 to 93,841 blocks lowers forced-DBO++ latency by 16.599% (95% interval $[15.889, 17.193]$). With allocation fixed, the profile effect is only 0.118%, with an interval crossing zero. The allocation retained after profiling, rather than the profiling workload alone, accounts for the large latency contrast.

The same pattern appears with unrelated memory allocations. Holding an equivalent number of active bytes outside the KV tensor adds 19.873% latency. Allocating that memory before or after compilation gives penalties of 19.834% and 20.014%, respectively. Freeing it restores near-control latency whether the allocator cache is retained or released (Supplementary Table 13). The early and late allocations are similar in effect, although they do not satisfy the preregistered strict equivalence bound. These controls identify live allocation as the relevant state without requiring a change to the visible KV tensor.

Runtime observations and an intervention connect this state to allocation handling. Inside vLLM, the absent, live and allocated-then-freed conditions record 1,157, 2,981 and 1,140 allocator retries, respectively. Live allocation adds 24.104% latency ($[22.237, 26.026]\%$); freeing it returns latency to the equivalence region at -0.151% ($[-0.518, 0.202]\%$). Retries occur in the fast controls too, but are more frequent in the slower live-allocation condition.

The intervention changes subsequent allocation handling while preserving existing allocations and all 93,841 KV blocks. Measured retries fall to zero, and the live-allocation penalty falls from 24.596% to -0.002% ($[-0.066, 0.091]\%$; Figure 2b). Selected tokens and recorded top-five values match. A separate fixed-startup path comparison measures a 23.965% penalty for overlapping execution, compared with 0.002% for unsplit execution and -0.023% for serial splitting. Both serial controls satisfy the registered equivalence bounds and record no retries. Their selected tokens agree across paths; top-five numerical values differ in some cases.

Together, the allocation, lifetime and path controls explain why identical batches can respond differently to engine configuration. The intervention also changes subsequent allocation behavior, so its effect encompasses more than a count of retries. The practical performance result comes from a separate matched comparison: with KV allocation fixed at 93,841 blocks, DBO++ reduces latency by 2.073% ($[1.888, 2.283]\%$), with positive means for every workload and process repetition. This net gain and the 24.6% within-path allocation penalty measure different aspects of the system.

4.4 Configuration Improves Held-Out Path Selection

Adding configuration to a workload-based rule preserves useful overlap choices while eliminating two harmful choices in the original held-out evaluation. Table 1 compares rules on 16 workload–configuration cases, with five paired process repetitions per forced action. All rules choose between the same OFF and ON paths. The token-only rule uses the workload interval from Section 3.3 without its capacity condition. The offline best choice provides the per-case reference.

DBO++ and the token-only rule retain 88.97% of the available positive benefit. DBO++ makes no harmful selections, compared with two for the token-only rule and four for always ON. Mean regret, normalized by each case's OFF latency, falls from 1.464% to 0.352%. The two metrics capture complementary behavior: benefit retention measures positive opportunities, while harmful-selection counts identify avoidable slowdowns. Always ON retains every positive opportunity but also selects every harmful case.

The $6 \times 3{,}412$ and $8 \times 2{,}559$ batches show why the capacity input matters: each keeps the same shape but changes its preferred path across configurations (Figure 3). Workload size still matters. A 21,512-token batch is slower with overlap even below half of the higher capacity; the upper token bound rejects it. The conservative lower bound also rejects two beneficial cases. Benefit retention has interval $[88.06, 89.94]\%$, and regret has interval $[0.318, 0.384]\%$. The metadata-only decision costs $2.269\ \mu$s at p99 over 100,000 CPU calls. All selected outputs match in this evaluation.

Table 1: Execution selection on 16 held-out Qwen3-32B TP4/H200 cases. All rules choose between the same OFF/ON paths. DBO++ retains the token-only rule's positive gains while avoiding its two harmful choices.

Rule	ON /16	Benefit retained (%) ↑	Harmful ON /4 ↓	Mean regret (%) ↓
Always OFF	0	0.00	0	3.190
Always ON	16	100.00	4	2.658
Token-only rule	12	88.97	2	1.464
DBO++ rule	**10**	**88.97**	**0**	**0.352**
Offline best choice	12	100.00	0	0.000

Benefit retained counts positive savings, without deducting harmful choices. Regret is excess latency over the offline choice, normalized by each case's OFF latency and averaged equally. All selected outputs match in this evaluation; independent follow-up failures are reported in Section 4.4.

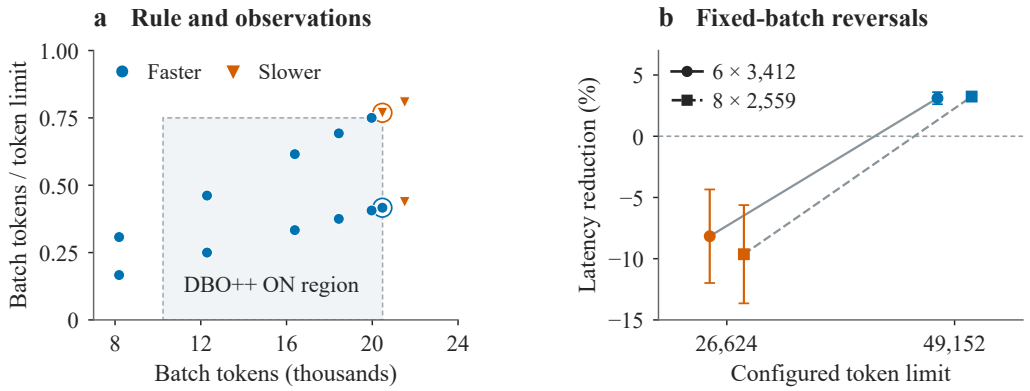

Figure 3: Configuration provides information beyond input size. (a) All 16 held-out cases: circles indicate faster overlap and triangles slower overlap. Shading shows the frozen rule; ringed locations contain two workloads. (b) Two fixed 20,472-token batches change their preferred path across configurations. Error bars are paired-process 95% t intervals over five repetitions.

The same rule transfers imperfectly to new capacities of 24,576 and 30,720. It selects three cases with changed outputs and two of six harmful cases whose outputs remain exact. Learned global and per-capacity intervals also select changed outputs (Supplementary Table 14). This pattern makes calibration part of the selection problem: the original capacity condition is informative, but its numerical boundary is specific to the measured regime. The independent results are reported alongside the original result rather than used to refit its rule.

4.5 Long Inputs Show the Most Consistent Prefill Gains

Large input batches yield consistent gains across the measured model/TP settings, whereas small and intermediate shapes are more heterogeneous. Figure 4 covers every TP2/TP4 position in the four-model grid: 275 numerical comparisons and five pairs with execution failures. Prompt lengths span 512–32,768 tokens and batch sizes span 1–16. The full-precision results, absolute large-input timings and failure details are in Supplementary Tables 5, 6 and 8.

In the original 255-pair survey, which includes 23 TP1 controls, 66 comparisons reduce median latency by more than 5%, 17 increase it by more than 5%, and 172 lie between these bounds. Llama's means are positive at both TP degrees, Qwen3.5-27B's are negative, and Gemma changes sign. The TP1 control stays close to zero (-1.62% to $+0.79\%$). These observations motivate examining the workload region rather than assigning one average speedup to a model or TP degree.

The large-input extension tests five shapes containing 131,072–524,288 tokens and one 8,192-token control in each model/TP setting. All 35 large shapes with two successful arms favor DBO, with reductions of 5.23–8.52% at TP4 and 0.58–3.69% at TP2 in the original readings. The eight small controls range from -0.44% to $+0.17\%$. The coherent large-input pattern contrasts with the sign changes at smaller shapes. These are eager-prefill observations under the survey's threshold and minimum-length settings, rather than tests of the calibrated selector.

Matched process repetitions confirm that the Qwen TP2 benefit persists with equal memory budgets. The original six Qwen TP2 extension pairs used budgets of 0.85 for baseline and 0.90 for DBO. Table 2 repeats those shapes with 0.85 on both paths and three independent process pairs. All five large shapes improve in every repetition, with reductions in mean process medians of 0.85–3.13%. The small control averages essentially zero. At the same total input size, the shorter-prompt, larger-batch shape gains more: $16 \times 8{,}192$ exceeds $8 \times 16{,}384$, and $16 \times 16{,}384$ exceeds $8 \times 32{,}768$. Benefit also falls at the largest measured shape. The advantage of the larger batch is 0.91 percentage points at 131K tokens and 0.89 points at 262K tokens. Increasing input size from 131K to 524K does not produce an increasing speedup: the maximum occurs at the intermediate $16 \times 8{,}192$ shape. This pattern identifies batch shape, alongside total tokens, as a useful variable for execution selection.

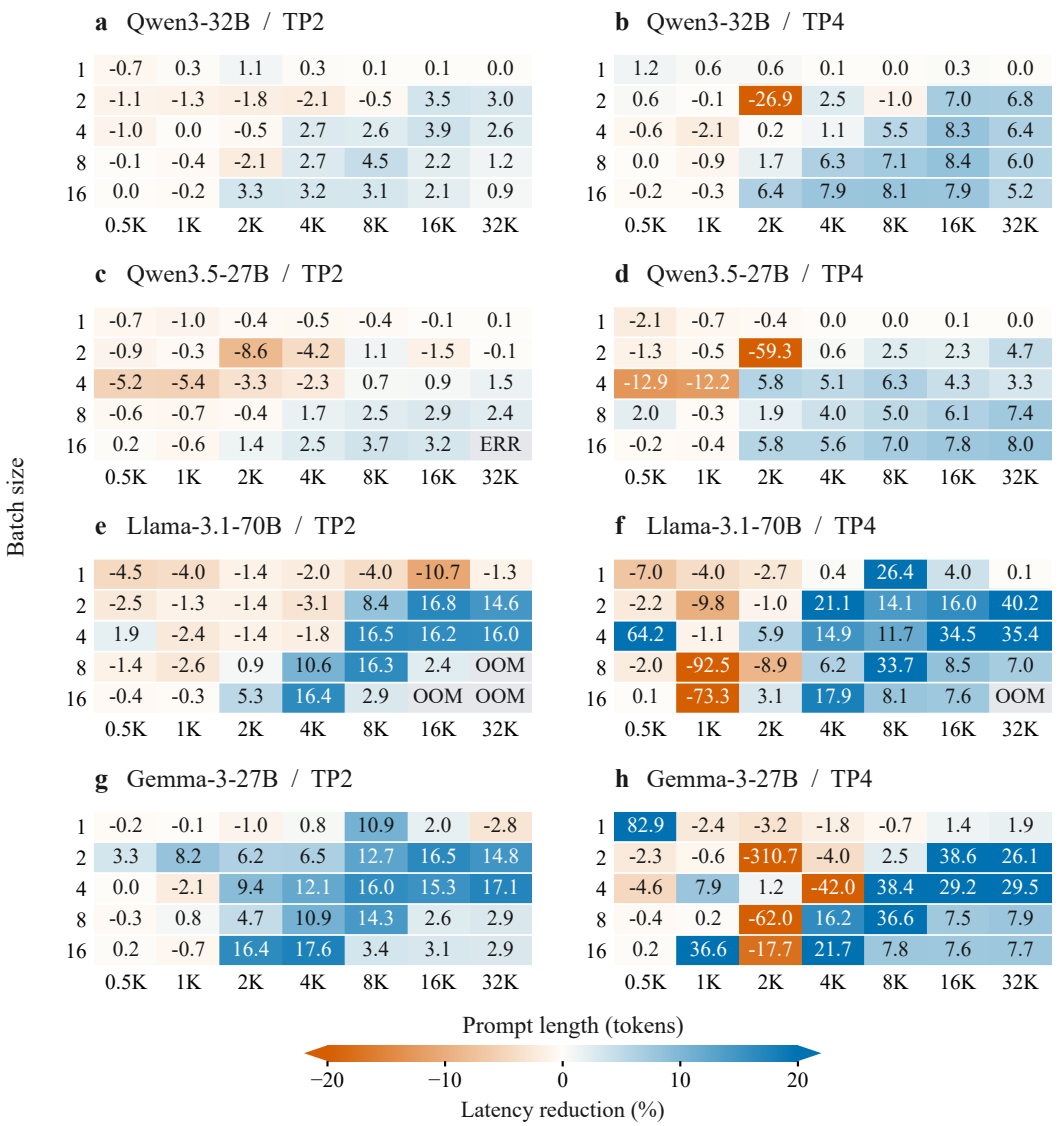

Figure 4: Complete TP2/TP4 prefill grid. Cells report latency reduction in percent; blue indicates faster DBO++, orange slower. The linear color scale saturates at $\pm 20\%$, while labels retain all extremes. OOM and ERR denote recorded out-of-memory and illegal-memory-access failures, respectively. The six additional Qwen3-32B TP2 cells use the matched repetitions in Table 2; other cells report the original per-process medians. Absolute timings, original readings and execution settings are retained in the supplement.

Table 2: Qwen3-32B TP2 with equal memory budgets of 0.85. Timings are means $\pm$ sample standard deviations of three process medians. Speedup is baseline/DBO.

Shape	Input tokens	Baseline (ms)	DBO (ms)	Speedup	Reduction (%)
16×512	8,192	479.0 ± 1.6	479.0 ± 0.7	1.0000	+0.001
$8 \times 16{,}384$	131,072	$9{,}189.5 \pm 22.7$	$8{,}985.2 \pm 14.0$	1.0227	+2.223
$16 \times 8{,}192$	131,072	$8{,}298.1 \pm 9.2$	$8{,}038.5 \pm 11.4$	1.0323	+3.129
$8 \times 32{,}768$	262,144	$21{,}973.6 \pm 52.9$	$21{,}710.4 \pm 56.4$	1.0121	+1.198
$16 \times 16{,}384$	262,144	$18{,}307.1 \pm 40.7$	$17{,}925.5 \pm 26.6$	1.0213	+2.084
$16 \times 32{,}768$	524,288	$43{,}840.2 \pm 91.7$	$43{,}467.3 \pm 82.2$	1.0086	+0.851

Startup memory demand limits the largest Llama configurations. Llama has baseline startup OOM at TP2 for $8 \times 32{,}768$, $16 \times 16{,}384$ and $16 \times 32{,}768$, and at TP4 for $16 \times 32{,}768$. DBO completes three of these four pairs; the largest TP2 shape has a recorded DBO startup OOM as well. The remaining failed pair, Qwen3.5-27B TP2 at $16 \times 32{,}768$, reports CUDA illegal memory access on the baseline side and a completed DBO run. The supplement records allocation requests, free memory and the surviving arm's latency. These outcomes establish feasibility under the tested launch settings; no speedup is assigned to a failed pair.

Output comparisons are reported separately from timings. In eight TP4 batches containing 43 requests per model, the first output token agrees for 42 Qwen requests and all 43 Llama requests. Qwen's mismatch occurs in the second request of the $4 \times 4{,}096$ batch. Complete sequences agree in 36 and 32 requests, respectively; the remaining divergences begin after prefill (Supplementary Table 10). The timing survey and these output checks have different coverage, so the table reports their measured agreement directly.

Parameter studies reinforce the importance of the execution regime: no tested threshold dominates every shape across 860 recorded calls. At $2 \times 16{,}384$, all tested thresholds reduce latency by 21.91–29.80% against the no-DBO control, while other shapes change sign. In the separate resource study, every nonzero requested communication-SM budget is slower than the zero-budget DBO control by 1.73–30.08%. Online serving also differs from single-token prefill; the recorded Llama TP4 configurations lose 16.79–82.74% throughput while changing runner and output work. The supplement retains these results and the separate common-reference studies, allowing the prefill patterns to be read alongside the conditions under which they change.

5 Discussion

The central systems implication is that engine configuration forms part of the execution context. A batch-token limit can change startup memory allocation and, through that state, the relative cost of two paths processing the same input. A selector based on workload alone misses this dependence. The successful held-out comparison shows the value of adding configuration, while the independent failures show that calibration must cover both performance and output behavior in the intended regime.

The allocation controls further distinguish cache capacity from allocation behavior. Reducing the retained cache by 1,201 blocks, or 1.26%, changes forced-overlap latency by 16.60%, even though the input is unchanged. The effect is concentrated in overlap rather than serial execution. The subsequent intervention removes the live-allocation penalty while retaining the existing cache. These results point to two separate design choices: how much memory to reserve for requests, and how to manage temporary allocations while their computation and communication overlap.

The measurements also separate three practical questions: whether overlap reduces latency, whether a large batch can execute, and whether its outputs agree with the reference. The matched fixed-cache and repeated TP2 studies answer the first; the long-input launch records answer the second; the explicit output checks address the third. The broad survey measures eager prefill through one output token. Extending these results to optimized compiled execution or online serving requires accounting for runner choice, KV capacity and request scheduling. Direct comparisons with complete systems such as TokenWeave and NanoFlow remain outside the present evaluation.

6 Conclusion

DBO++ extends dual-batch execution to dense tensor-parallel prefill. Controlled experiments establish that engine configuration can reverse overlap's benefit for a fixed batch and identify live allocation and subsequent allocation handling as contributors. With cache allocation fixed, the controlled comparison yields a 2.1% latency reduction; matched TP2 repetitions recover 0.85–3.13% reductions on long inputs. Configuration improves held-out execution choices in the calibrated regime, and the broader measurements locate consistent gains at large input sizes. These findings make the engine state a concrete part of deciding when to overlap, alongside the workload and the output behavior of the chosen path.

AI USE STATEMENT

Generative AI tools assisted with research planning and experiment design, implementation, evidence reconciliation and interpretation, scientific plotting, manuscript drafting, translation and editing. Numerical tables and experimental figures were produced from recorded measurements. AI assistance also supported consistency checks between the source records, analysis scripts and manuscript. The authors take responsibility for the final text, claims, code and experimental evidence.

REPRODUCIBILITY STATEMENT

Section 3 and Supplementary Appendix A describe the execution paths, stream ordering and empirical admission rule. The evaluation specifies measurement units and independent repetitions. The supplementary material provides the complete prefill grids, absolute long-input timings, matched repetitions, output comparisons, failure evidence, parameter studies and serving controls. Figure captions and table notes identify reference paths and uncertainty estimates; the measurement-scope table distinguishes the validation available for each study.

REFERENCES

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*, pp. 117–134. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

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.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, 2022. doi: 10.52202/068431-1189.

Jiangsu Du, Jinhui Wei, Jiazhi Jiang, Shenggan Cheng, Dan Huang, Zhiguang Chen, and Yutong Lu. Liger: Interleaving intra- and inter-operator parallelism for distributed large model inference. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pp. 42–54, 2024. doi: 10.1145/3627535.3638466.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025. URL https://arxiv.org/abs/2503.19786.

Raja Gond, Nipun Kwatra, and Ramachandran Ramjee. TokenWeave: Efficient compute-communication overlap for distributed LLM inference. In *Proceedings of the Ninth Conference on Machine Learning and Systems*, 2026. URL https://arxiv.org/abs/2505.11329v5.

Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, and Yuxiong He. DeepSpeed-FastGen: High-throughput text generation for LLMs via MII and DeepSpeed-Inference. *arXiv preprint arXiv:2401.08671*, 2024. URL https://arxiv.org/abs/2401.08671.

Ke Hong, Xiuhong Li, Minxu Liu, Qiuli Mao, Tianqi Wu, Zixiao Huang, Lufang Chen, Zhong Wang, Yichong Zhang, Zhenhua Zhu, Guohao Dai, and Yu Wang. Efficient and adaptable overlapping for computation and communication via signaling and reordering. In *Proceedings of the 21st European*

Conference on Computer Systems, pp. 1894–1911, 2026. doi: 10.1145/3767295.3769370. URL https://arxiv.org/abs/2504.19519.

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.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Llama Team. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using Megatron-LM. *arXiv preprint arXiv:2104.04473*, 2021. URL https://arxiv.org/abs/2104.04473.

Yi Pan, Yile Gu, Jinbin Luo, Yibo Wu, Ziren Wang, Hongtao Zhang, Ziyi Xu, Shengkai Lin, Baris Kasikci, and Stephanie Wang. DynaFlow: Transparent and flexible intra-device parallelism via programmable operator scheduling. In *Proceedings of the Ninth Conference on Machine Learning and Systems*, 2026. URL https://arxiv.org/abs/2605.21603.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pp. 8026–8037, 2019.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, pp. 118–132, 2024. doi: 10.1109/ISCA59077.2024.00019.

Suchita Pati, Shaizeen Aga, Mahzabeen Islam, Nuwan Jayasena, and Matthew D. Sinclair. T3: Transparent tracking & triggering for fine-grained overlap of compute & collectives. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1146–1164, 2024. doi: 10.1145/3620665.3640410.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vAttention: Dynamic memory management for serving LLMs without PagedAttention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2025. URL https://arxiv.org/abs/2405.04437.

Xinwei Qiang, Yue Guan, Zhengding Hu, Keren Zhou, Yufei Ding, and Adnan Aziz. Syncopate: Efficient multi-GPU AI kernels via automatic chunk-centric compute-communication overlap. In *20th USENIX Symposium on Operating Systems Design and Implementation*, pp. 331–347. USENIX Association, 2026. URL https://www.usenix.org/conference/osdi26/presentation/qiang.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

vLLM Project. Dual batch overlap. https://docs.vllm.ai/en/stable/design/dbo/, 2026. Accessed 2026-07-28.

Bin Xiao and Lei Su. ISO: Overlap of computation and communication within seqenence for LLM inference. *arXiv preprint arXiv:2409.11155*, 2024. URL https://arxiv.org/abs/2409.11155.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation*, pp. 521–538. USENIX Association, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Muru Zhang, Mayank Mishra, Zhongzhu Zhou, William Brandon, Jue Wang, Yoon Kim, Jonathan Ragan-Kelley, Shuaiwen Leon Song, Ben Athiwaratkun, and Tri Dao. Ladder-residual: Parallelism-aware architecture for accelerating large model inference with communication overlapping. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75779–75792, 2025. URL https://proceedings.mlr.press/v267/zhang25bg.html.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000.

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.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation*, pp. 193–210. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

Kan Zhu, Yufei Gao, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Tian Tang, Qinyu Xu, Zihao Ye, Keisuke Kamahori, Chien-Yu Lin, Ziren Wang, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. NanoFlow: Towards optimal large language model serving throughput. In *19th USENIX Symposium on Operating Systems Design and Implementation*, pp. 749–765. USENIX Association, 2025. URL https://www.usenix.org/conference/osdi25/presentation/zhu-kan.