FLEXBI: REGAINING EXECUTION FLEXIBILITY FOR BATCH-INVARIANT LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Batch-invariant (BI) inference preserves bitwise-identical results for each request regardless of its surrounding batch. Existing BI implementations couple this numerical guarantee to fixed physical schedules, limiting their ability to exploit changing concurrency and prefix sharing. FLEXBI regains execution flexibility by separating a fixed request-local numerical contract from adaptive physical scheduling. FLEXTILE regains dynamic GEMM output tiling by keeping each output's reduction trace invariant across tile changes. FLEXLOAD uses partition-invariant sharing to share identical physical key-value (KV) page loads while preserving each query's logical reduction. In online serving across two models and two datasets, FLEXBI delivers 25–33% higher output throughput and 16–32% lower P99 time to first token than same-engine BI in vLLM and SGLang.

1 INTRODUCTION

Batch-invariant (BI) inference for a large language model (LLM) requires a request's numerical result to remain bitwise identical as its surrounding batch changes, with the model and execution platform fixed (He & Thinking Machines Lab, 2025). In online serving, BI must hold as continuous batching admits and retires other requests and prefix caching changes which requests share physical KV pages (Yu et al., 2022; Kwon et al., 2023; Zheng et al., 2024). To maintain that guarantee as computation shapes change, BI preserves the request's floating-point reduction order, keeping its autoregressive response and task outcome reproducible (He & Thinking Machines Lab, 2025; Yuan et al., 2025). Request-level reproducibility supports consistent reinforcement-learning rollouts (Zhang et al., 2026), reduces confounding in failure attribution for multi-agent systems (Chen et al., 2026), and enables bitwise verification of local LLM inference (Molina, 2026).

However, enforcing BI can limit execution flexibility when an implementation freezes physical choices that a serving engine would otherwise adapt. Existing BI implementations make the reduction structure of numerically sensitive operators independent of batch composition (He & Thinking Machines Lab, 2025; vLLM Project, 2026; SGLang Project, 2026; Gond et al., 2026). General matrix multiplication (GEMM) follows a fixed accumulation order, and Attention uses stable partitions and merge orders for each query. The implementations examined in this work also retain fixed output mappings and separate KV loads across queries. We refer to this combination as *fixed-schedule BI execution*. Coupling numerical invariance to these physical choices limits the serving engine's ability to absorb changes in concurrency and prefix sharing (Figure 1).

To address this limitation, we identify a key insight: a fixed numerical computation admits multiple execution schedules, allowing batch invariance and execution flexibility to coexist. Preserving each request's operands, arithmetic order, and rounding behavior leaves two forms of execution flexibility available. For GEMM, BI constrains the floating-point reduction trace of each output element, rather than the complete physical execution plan. Independent outputs can therefore be retiled and redistributed across GPU thread blocks while each output retains the same ordered arithmetic operations and rounding points. A fixed output mapping leaves this freedom unused as batch size changes, leading to masked rows or insufficient parallel work. For Attention, queries can share loads of identical read-only KV data while retaining separate computations. Paged and radix caches expose this opportunity through shared physical pages (Kwon et al., 2023; Zheng et al., 2024), yet separate loads across queries repeat data movement. These two costs motivate adapting

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.

the placement of independent computation and the movement of shared data within a fixed numerical contract.

Accordingly, we present FLEXBI, which regains execution flexibility by separating a fixed numerical program from a dynamic physical work map. A numerical contract fixes the request-local arithmetic that determines bits, while the runtime rebuilds only the mapping of work and read-only data to GPU resources. FLEXTILE applies this separation to GEMM through reduction-invariant tiling. It separates the output tile from the per-output reducer, allowing tile dimensions and cooperative thread array (CTA) assignments to change while each output retains the same operands, accumulation chain, and rounding points. FLEXLOAD applies the same separation to Attention through partition-invariant sharing. It preserves each query's logical reduction graph and private state while grouping rows that read identical physical KV pages into shared delivery tasks. This operator-level design makes FLEXBI reusable across model architectures and serving engines without model-specific scheduling rules.

We integrate FLEXBI into vLLM and SGLang and evaluate online serving, controlled scaling, operator-level adaptation, and runtime overhead. Across two models and two datasets, FLEXBI improves output throughput and P99 time to first token over same-engine BI in online serving. Controlled workloads and operator ablations expose the gains from each mechanism. In the measured vLLM decode workloads, adaptation overhead is 0.44–0.57% of a complete step.

This work makes three contributions:

- We frame batch-invariant LLM serving as adaptive scheduling within a request-local numerical contract: GEMM fixes each output's reduction trace while leaving physical placement and read-only data movement flexible.
- We design reduction-invariant tiling in FLEXTILE to adapt GEMM output tiles while preserving each output's reducer. We design partition-invariant sharing in FLEXLOAD to deliver shared KV under fixed query-local reductions. Operator ablations show 1.38–$1.74\times$ GEMM and 1.38–$1.50\times$ attention speedups over native BI, with bitwise-identical outputs.
- We evaluate FLEXBI in vLLM and SGLang across two models and two datasets. Online serving improves output throughput by 25–33% and reduces P99 time to first token by 16–32% over same-engine BI. In the measured vLLM decode workloads, adaptation overhead is 0.44–0.57% of a complete step.

2 BACKGROUND AND MOTIVATION

2.1 BATCH INVARIANCE AS A SERVING GUARANTEE

Batch-invariant (BI) inference makes a request's numerical result independent of its execution context (He & Thinking Machines Lab, 2025). Dynamic batching changes tensor shapes (Yu et al., 2022) and can select different reduction schedules (He & Thinking Machines Lab, 2025; Yuan et al., 2025). Even deterministic kernels can then disagree bitwise because floating-point addition is nonassociative (Goldberg, 1991). Let $F_r(x_r; c)$ denote the effective logits and persistent KV state of request r with logical input x_r, aligned by logical token position. The allowed contexts $\mathcal{C}$ vary cobatches, request positions, cache states, page layouts, and legal execution schedules while holding the model and execution platform fixed. Request-level BI requires

$$\mathrm{bits}(F_r(x_r; c_1)) = \mathrm{bits}(F_r(x_r; c_2)) \quad \forall c_1, c_2 \in \mathcal{C}.$$

This guarantee fixes numerical results, leaving work placement and read-only data movement free.

Existing implementations obtain reproducibility by stabilizing reductions or execution shapes. BI kernels fix the ordered K reduction in GEMM and the KV partitions and merge order in attention (He & Thinking Machines Lab, 2025). Tree Based Invariant Kernels (TBIK) extends deterministic reductions across tensor-parallel sizes (Zhang et al., 2026). LLM-42 targets token-level determinism through verified speculation (Gond et al., 2026). Token agreement alone is weaker than this request-level bitwise guarantee. We study fixed-schedule BI execution, where stable reductions are coupled to fixed GEMM output tiling and separate per-query KV loads. The following sections examine which physical choices can vary while preserving request-level BI.

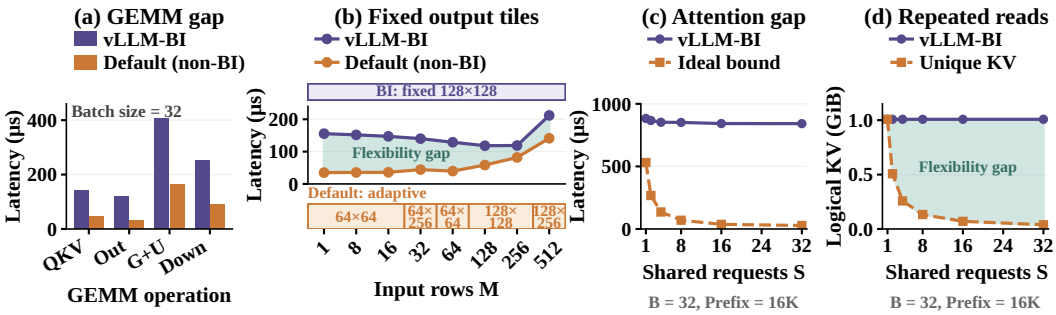


Figure 1: GEMM latency under fixed and adaptive output tiling in (a) and (b). BI attention latency and logical versus unique KV demand as physical prefix sharing increases in (c) and (d). Shading marks the flexibility gap targeted by FLEXBI.

2.2 GEMM OUTPUT TILING

Fixed output tiling constrains GEMM more than BI requires. Consider $C = AB$, where $A \in \mathbb{R}^{M\times K}$ and $B \in \mathbb{R}^{K\times N}$. Each output C_{ij} has an independent floating-point reduction along K, and its bits depend on the ordered accumulation. Output tiling instead groups independent elements of C and maps them to GPU resources. BI therefore requires a stable reduction for each output, while the output tile remains free to change.

Dynamic serving changes M as requests enter, finish, or move between prefill and decode. A large fixed tile masks many rows at small M and exposes fewer output tiles for parallel execution; coarse output tiling can leave GPU resources underutilized (Osama et al., 2023). Adapting the tile can reduce this wasted work without changing any output's ordered K reduction. Existing fixed-schedule BI kernels bind the reduction and output tile in one configuration, so the physical schedule cannot follow the active shape.

The Qwen3-32B (Yang et al., 2025a) experiment in Figure 1(a) shows the fixed-schedule gap across four principal linear layers. For the query-key-value (QKV) projection, Figure 1(b) shows how the gap changes with M. The vLLM-BI path keeps a 128×128 output tile, while the default non-BI path adapts to the active shape. At $M = 32$, the two paths take 140.1 and 44.3 μs. Changing only the output tile within the BI kernel preserves bitwise-identical outputs across all evaluated M values. This result motivates FLEXTILE: adapt the output work map while keeping each output's reduction fixed.

2.3 SHARED-PREFIX KV LOADS

Per-query attention leaves shared KV loads unexploited. Prefix sharing exposes the same distinction in data movement. System prompts, few-shot examples, parallel samples, and branches in tree-structured generation create batches of requests with identical prefix tokens (Juravsky et al., 2024). PagedAttention and RadixAttention represent this common prefix with shared physical KV pages and avoid repeated prefix prefill (Kwon et al., 2023; Zheng et al., 2024). These mechanisms establish shared storage, while decode attention still determines how each query consumes that storage.

With logical query work fixed, the experiment in Figure 1(c) varies the number of requests that share each prefix copy. The BI attention latency remains near 0.85 ms even when all 32 requests share the prefix. The KV-demand comparison in Figure 1(d) identifies repeated loads behind the flat latency response. Independent scans issue 1.008 GiB of logical KV demand, although the 32 requests reference only 0.039 GiB of unique KV data. This gap quantifies the load-sharing opportunity left unused by per-query attention.

The numerical constraint lies in the softmax reduction, not in KV-load ownership. BI must preserve each query's partitions, softmax state, and merge order. Hydragen and PAT exploit shared prefixes to reduce repeated KV loads (Juravsky et al., 2024; Yi et al., 2026). PAT packs queries by shared prefix and merges partial attention results from workload-dependent KV partitions (Yi et al., 2026). Such

partitioning does not guarantee BI: changes in the co-batch or cache state can change the softmax reduction structure even when a request's logical input is unchanged. Current BI execution avoids that dependence by retaining separate per-query scans. Immutable KV data can serve multiple query-local reductions without merging their numerical state. Fixed-schedule BI couples private numerical state to private data movement, discarding reuse as concurrent consumers of a cached prefix grow.

2.4 EXECUTION FLEXIBILITY

One numerical computation admits multiple physical schedules. The two observations follow from the same mismatch. Existing BI paths make the physical schedule invariant where only the request-local numerical computation must remain invariant. GEMM needs a stable reduction for each output, not a fixed placement of independent outputs. Attention needs stable per-query partitions and merge order, not repeated movement of identical read-only KV data.

FLEXBI makes this boundary explicit at the serving runtime. A request-local numerical contract fixes the decisions that determine a request's bits, while a step-local physical work map follows the current batch and page-sharing graph. FLEXTILE regains GEMM tiling flexibility by changing output tiles and GPU resource allocation while preserving every output's ordered reduction. FLEXLOAD regains KV-load flexibility by grouping queries that reference identical physical pages while preserving each query's logical attention partitions, softmax state, and merge order. Section 3 defines both contracts and the schedules they admit.

3 FLEXBI DESIGN

3.1 OVERVIEW

FLEXBI regains execution flexibility for batch-invariant LLM serving by adapting GPU execution at every scheduler step without changing the request-local arithmetic that determines output bits. A numerical contract defines the fixed *numerical program*: operand identities, arithmetic dependencies and order, precision, masks, epilogues, and rounding points. A workload-dependent *physical work map* assigns independent output work and immutable data delivery to GPU resources. FLEXTILE preserves each GEMM output's reducer while rebuilding its output mapping for the active shape. FLEXLOAD preserves each query's reduction graph and private state while rebuilding shared KV delivery from the active page graph.

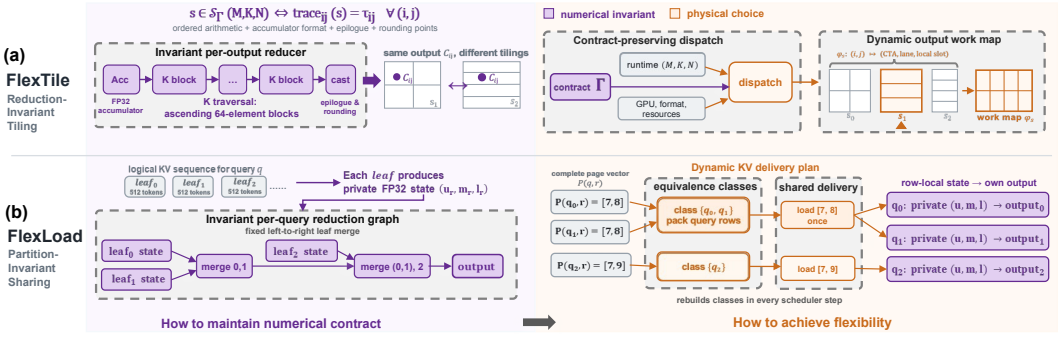


Figure 2: (a) FLEXTILE changes the output work map ϕ_s within $\mathcal{S}_\Gamma(M, K, N)$ while preserving each C_{ij} reducer. (b) FLEXLOAD preserves each query's reduction graph and private state while rebuilding KV delivery from complete physical page vectors.

Both panels of Figure 2 place invariant numerical structure beside a rebuilt physical work map. Each descriptor identifies its numerical contract. The physical work map can therefore change while the request-local program remains fixed. Structural admission defines legal descriptors. Compiled bitwise probes test their kernels. Uncovered shapes use a same-contract fallback.

3.2 FLEXTILE: REDUCTION-INVARIANT TILING

Consider $C = AB$, where $A \in \mathbb{R}^{M\times K}$ and $B \in \mathbb{R}^{K\times N}$. Each C_{ij} is independent of the other output elements. Its bits depend on its floating-point reduction along K. BI therefore constrains the reduction trace of each output element, not the complete physical execution plan of the GEMM. Fixed-schedule BI obscures this boundary by encoding the reduction and the output tiling in one kernel configuration. FLEXTILE regains tiling flexibility by keeping the per-output reducer invariant across changes to tile dimensions and output placement. Figure 2(a) places the invariant per-output reducer on the left and the shape-dependent output work map on the right.

Let τ_{ij} be the numerical program that contract Γ assigns to C_{ij}. It contains the operand identities, ordered arithmetic operations, accumulator format, split structure, epilogue, output conversion, and rounding points. A physical schedule s supplies a work map $\phi_s : (i, j) \mapsto (\text{CTA}, \text{lane}, \text{local slot})$ and a data-staging plan. We admit s under Γ only when projecting its execution onto the arithmetic events for every output recovers the contracted program:

$$s \in \mathcal{S}_\Gamma(M, K, N) \quad \Longleftrightarrow \quad \text{trace}_{ij}(s) = \tau_{ij} \quad \forall (i, j).$$

This condition defines a family of legal schedules rather than one BI configuration. Every member computes the same numerical program for each output, although its work map can differ.

Invariant per-output reducer. FLEXTILE represents the contracted numerical program as a reducer that each output tile instantiates. The evaluated BF16 reducer visits 64-element K blocks in ascending order, updates one FP32 accumulation chain per output, applies any bias in FP32, and casts once to the output format. A single CTA writes each output. Tile changes preserve the contracted reduction and rounding points. The schedule descriptor contains only physical choices. The contract fixes the kernel family, K block size, accumulation order, precision, split structure, and epilogue.

Dynamic output work map. For the active (M, K, N) shape, FLEXTILE partitions the logical output domain $[M] \times [N]$ into disjoint tiles and maps those tiles to CTAs. The descriptor can change the M/N tile, row grouping, warp count, pipeline depth, and assignment of tiles to CTAs. Boundary predicates remove out-of-range elements without adding numerical dependencies between valid outputs. Pipeline stages may regroup or overlap loads only when arithmetic events reach each accumulator in the contracted order. The resulting work map can reduce work on out-of-range output rows, expose more output tiles, or change resource use while the per-output reducer remains intact.

BI across tile changes. Consider a target row moved to a different batch position, possibly under a different output tile. For each of its output columns, the new work map supplies the same logical operands to the same ordered reducer. Starting from the same accumulator value, each contracted update therefore produces the same bits, followed by the same epilogue and output cast. Tile changes alter which independent outputs execute together while preserving the numerical program of every valid output. This recovers dynamic output tiling within $\mathcal{S}_\Gamma$. Every admitted schedule preserves BI independently of the latency policy used to choose it.

Validation and fallback. Structural admission requires every descriptor to preserve the contracted reducer and to cover each valid output exactly once. Compiled bitwise probes validate generated kernels on the tested inputs. Uncovered shapes use a same-contract fallback. Appendix B.3 gives the admission and dispatch procedure. Offline latency calibration selects admitted schedules for a shape-indexed runtime lookup.

3.3 FLEXLOAD: PARTITION-INVARIANT SHARING

Exact tiled attention maintains online-softmax state while it scans the score matrix (Dao et al., 2022). Existing BI paths bind two decisions together: how each query reduces its logical KV sequence and how the kernel fetches the physical pages that hold that sequence. Only the first decision determines the query's numerical trace. The logical KV partition must be independent of the current shared-prefix boundary because cache hits and co-batches can move that boundary. Identical immutable pages remain available for shared loading across queries. FLEXLOAD separates the query-local reduction graph from KV delivery adapted to the current page graph (Figure 2(b)).

Invariant per-query reduction graph. The attention contract defines a reduction graph in logical token coordinates, independent of page ownership and the current set of consumers. It divides each KV sequence into fixed logical leaves. Logical leaf r advances online softmax in a fixed token order and keeps private state $(\mathbf{u}_r, m_r, \ell_r)$, where m_r is the running maximum, ℓ_r is the normalizer, and $\mathbf{u}_r$ is the unnormalized value sum. The kernel merges leaves from left to right:

$$\begin{aligned} m &= \max(m_a, m_b), \\ \ell &= e^{m_a - m}\ell_a + e^{m_b - m}\ell_b, \\ \mathbf{u} &= e^{m_a - m}\mathbf{u}_a + e^{m_b - m}\mathbf{u}_b. \end{aligned}$$

Logical leaf boundaries, traversal order, partial precision, masks, and merge order are invariant to the co-batch, cache hit, and number of consumers.

Dynamic KV delivery plan. Let $P(q, r)$ denote the complete ordered vector of physical page identifiers that supplies logical leaf r to query row q. For each leaf, FLEXLOAD constructs the equivalence classes

$$q_a \sim_r q_b \quad \Longleftrightarrow \quad P(q_a, r) = P(q_b, r).$$

Rows in one class consume the same immutable KV elements. One physical tile can load a KV fragment and provide it to several row-local computations. The class changes as requests enter, leave, or acquire cache hits. The logical leaf r and its position in each query's reduction graph do not change.

A GPU planner rebuilds classes from page tables each step and packs their rows into physical tiles. It compares complete page vectors before sharing a load, grouping by physical identity rather than request labels or estimated prefix length.

State isolation under shared delivery. The grouped kernel shares operand delivery, not numerical state. Each row reads its own query, maintains its own $(\mathbf{u}, m, \ell)$ state, applies its own causal or sliding-window mask, and writes to its own output location. The shared KV fragment enters the same logical leaf computation that an independent load would have supplied. Request permutation, class size, and task order can therefore change the physical delivery plan while preserving every query's leaf program and fixed left-to-right merge. This is partition-invariant sharing: the sharing partition follows physical page identity, while the numerical partition remains fixed in logical coordinates.

Prefix caching and FLEXLOAD remove costs at different times. A prefix-cache hit shares storage and can skip historical prefill (Kwon et al., 2023; Zheng et al., 2024). FLEXLOAD acts during the current decode step, when several queries load the same resident pages. The same-contract per-query path keeps the logical reduction graph and disables only cross-request KV-load sharing.

3.4 CROSS-ENGINE SYSTEM INTEGRATION

Request-level BI. Within either engine, the operator contracts compose across layers and decode steps. For identical logical inputs and persistent state, each contracted operator produces identical output bits, while all other operators remain on that engine's BI paths. Across legal execution contexts, the final logits are therefore bitwise identical, and fixed token selection establishes the same token and persistent KV state for the next step.

Common integration boundary. We reuse FLEXBI's contracted operators across models and connect them to serving engines through adapters. FLEXTILE maps an active GEMM shape to an admitted schedule descriptor. FLEXLOAD maps sequence lengths, page tables, query metadata, and masks to a delivery plan. Each adapter retains its engine's scheduler, paged KV cache, and phase routing. The adapters replace only the contracted GEMM and single-token decode attention paths. Multi-token prefill remains on each engine's BI attention path, and both prefill and decode preserve the logical KV layout.

Graph-compatible dynamic scheduling. CUDA Graph replay fixes kernel topology and buffer addresses (NVIDIA, 2026), while scheduler steps can select different shape buckets and expose different physical page-sharing graphs. Before capture, each engine allocates planner and execution

workspaces at addresses reused by every replay. For FLEXTILE, each CUDA Graph bucket captures the admitted schedule for its matrix shape. Runtime bucket selection provides shape adaptation. For FLEXLOAD, the engine refreshes sequence lengths and page tables before replay. The GPU planner then rewrites the preallocated task buffers and device-side task count. The captured attention kernels consume these buffers without allocating memory, synchronizing with the CPU, or recapturing the graph. The cache manager retains every referenced page until the scheduler step completes. Appendix B gives the implementation details.

4 EVALUATION

We therefore ask four research questions (RQs): **RQ1: Online end-to-end serving.** Does FLEXBI improve P99 time to first token and completed output throughput under trace-driven arrivals? **RQ2: Controlled scaling.** How does the complete system respond as independent work and physical KV sharing change? **RQ3: Operator-level ablation.** How do FLEXTILE and FLEXLOAD perform in isolation across operator shapes from different model architectures, and which physical schedules do they select? **RQ4: Runtime overhead.** How much per-step time and workspace does this adaptation add?

4.1 EXPERIMENTAL SETUP

Environment. We evaluate vLLM v0.26.0 and SGLang v0.5.16 with PyTorch 2.11.0 and CUDA 12.8 on two A100-SXM4-80GB GPUs using TP2 and BF16. Qwen3-14B and StarCoder2-15B-Instruct-v0.1 provide full-context and sliding-window attention at a comparable model scale (Yang et al., 2025a; Lozhkov et al., 2024). Controlled experiments and component ablations use Qwen3-14B.

Baselines. We pair each FlexBI integration with its engine's unmodified BI path, denoted vLLM-BI and SGLang-BI (vLLM Project, 2026; SGLang Project, 2026). FlexBI-vLLM and FlexBI-SGLang preserve all other engine paths and replace only the contracted BI GEMM and decode attention operators. As an additional determinism-oriented control, we preserve TBIK's pinned row-parallel GEMM and tree all-reduce in a vLLM v0.26.0 port (Zhang et al., 2026).

Datasets. We use the Mooncake FAST'25 ToolAgent and Conversation traces (Qin et al., 2025). We freeze three non-overlapping StarCoder2 slices per workload, preserving source lengths, request order, and anonymous prefix relations. Both models and systems replay each slice; points average the three measurements. Appendices D.3 and D.2 detail the protocols and trace construction.

4.2 RQ1: ONLINE END-TO-END SERVING

RQ1 tests whether recovered execution flexibility improves service-level behavior as offered load changes. Figure 3 compares systems that replay identical trace-derived request streams under a matched end-to-end protocol.

Tail latency. FlexBI delays the sharp increase in P99 TTFT as offered load approaches the capacity region. At the highest measured rates, it reduces P99 TTFT by 16–32% relative to the same-engine BI baseline across both models and workloads. On StarCoder2 ToolAgent at 2.4 requests/s, for example, FlexBI-SGLang lowers P99 TTFT from 82.5 to 56.4 seconds. The paired curves remain close at low load and separate as concurrency grows, consistent with FlexBI using the additional output-remapping and KV-sharing opportunities to limit queue growth.

Output throughput. FlexBI also sustains more completed output work before throughput plateaus. At the highest measured rates, it delivers 25–33% higher output throughput than the same-engine BI baseline. For the same StarCoder2 ToolAgent point, FlexBI-SGLang reaches 207.0 output tokens/s, compared with 163.3 output tokens/s for SGLang-BI, and every request completes in all three sessions. The throughput gain holds in both serving engines and both workload traces, including full-context and sliding-window attention.

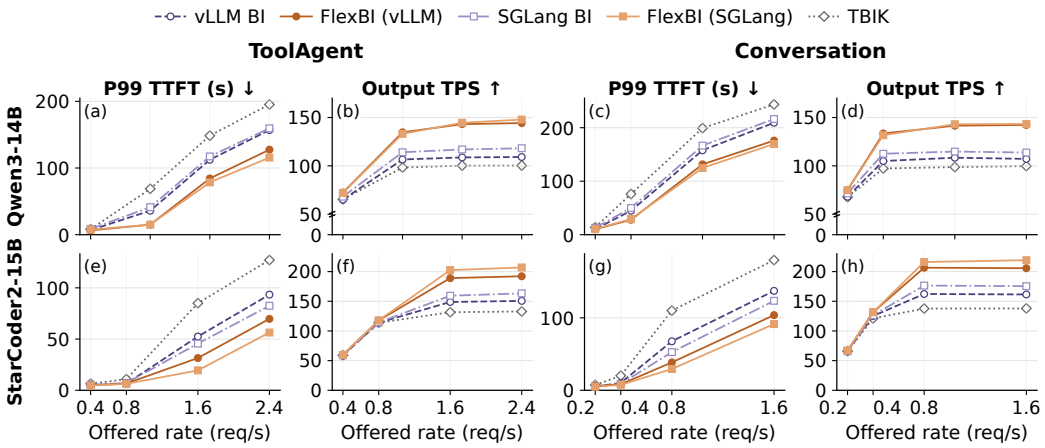


Figure 3: Online P99 time to first token (TTFT) and output tokens per second (TPS) across offered request rates. Rows separate the two models, and columns separate the two trace workloads.

4.3 RQ2: Scaling with work and sharing

RQ2 varies independent work and physical KV sharing with the complete engine active. Figure 4 compares each FlexBI integration with its same-engine BI baseline.

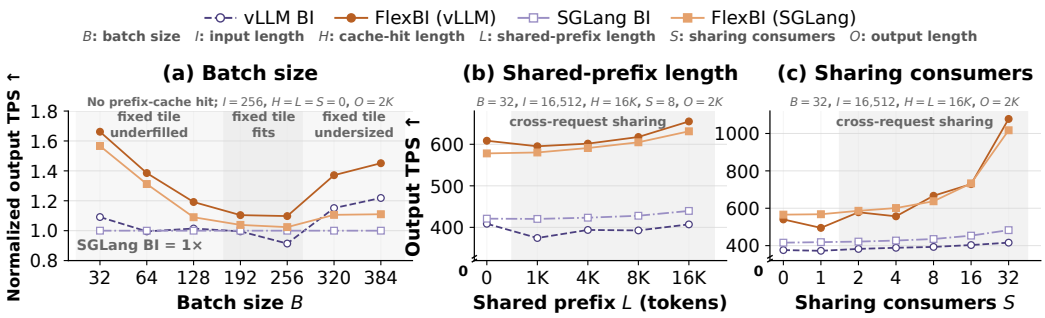


Figure 4: Controlled scaling. (a) Output tokens/s normalized to SGLang-BI at each batch size. (b–c) Absolute output tokens/s across shared-prefix lengths and consumer counts.

Independent work. FlexBI recovers throughput where the fixed GEMM output mapping fits the batch size poorly. A short, cache-free workload sweeps batch sizes through $B = 384$ to isolate the effect of independent work from prefix reuse. Both FlexBI integrations exceed their same-engine baselines throughout Figure 4(a). The gap narrows near $B = 192$–256, where the fixed tile is well matched, and grows when FlexTile must correct an underfilled or undersized tile.

Physical sharing. FlexBI retains its throughput advantage from short shared prefixes and widens it as sharing grows. At a fixed 16K-token cache hit, both complete integrations exceed their same-engine baselines even with only a 1K-token shared prefix in Figure 4(b). As S grows, FlexLoad shares each KV load across more requests. At $S = 32$, FlexBI-vLLM and FlexBI-SGLang reach 1,077.8 and 1,017.2 output tokens/s, respectively (Figure 4(c)).

4.4 RQ3: Operator-level ablation and adaptation

RQ3 benchmarks contracted GEMM and attention operators in isolation across ten dense and mixture-of-experts model configurations to test whether FlexTile and FlexLoad adapt beyond the two serving models (Figure 5). The GEMM sweep spans representative QKV projections and

feed-forward network (FFN) down projections across TP-local input and output widths, testing whether one shape-driven scheduler covers diverse architectures without model-specific rules.

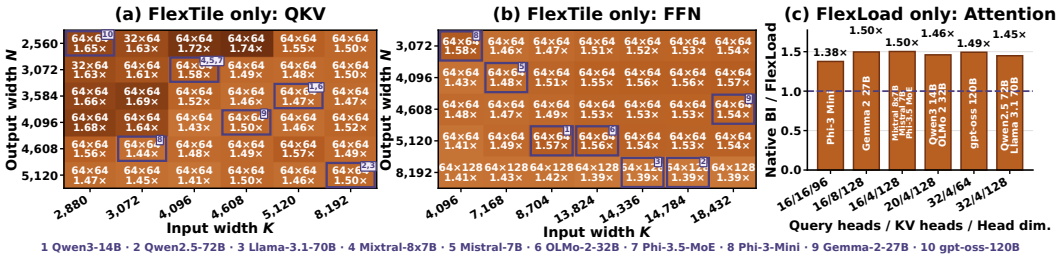


Figure 5: Operator ablation relative to native BI. In (a–b), FLEXTILE fixes the input rows at $M = 64$; the axes enumerate operator shapes (K, N), while each cell reports the selected $\mathrm{BM} \times \mathrm{BN}$ tile and speedup. Purple numbers index models. In (c), FLEXLOAD uses $B = 32$, with one shared group of eight requests and 24 independent requests.

FlexTile. Across the measured QKV and FFN shapes, FLEXTILE delivers 1.38–$1.74\times$ GEMM speedups over native BI with three selected output tiles (Figure 5(a–b)). The selected tile changes with the active operator shape, rather than the model that supplies the shape. Every output remains bitwise identical to native BI: tiles remap independent outputs while preserving each output's ordered K reduction.

FlexLoad. FLEXLOAD delivers 1.38–$1.50\times$ attention speedups over native BI when eight of 32 requests share a 16K-token prefix (Figure 5(c)). At the peak, physical sharing cuts task count by $1.27\times$. The matched FLEXLOAD and Canonical no-sharing paths produce bitwise-identical outputs and per-query state across ten model configurations, without model-specific paths. These are isolated operator measurements at fixed shapes. The matched online and controlled experiments above include the engine scheduler and unchanged operators, and therefore determine the end-to-end gain.

4.5 RQ4: RUNTIME AND WORKSPACE OVERHEAD

RQ4 quantifies the runtime and memory required to adapt the physical schedule. Table 1 profiles the complete decode step under workloads with and without physical KV sharing.

Table 1: Per-step runtime overhead.

Regime	Step (ms)	KV plan (μs, %)	State + merge (μs, %)	Added (μs, %)
Independent	32.65 ± 0.22	7.78 ± 0.00 (0.024)	137.2 ± 0.0 (0.420)	145.0 ± 0.0 (0.444)
16K, $S = 8$	48.95 ± 1.50	11.60 ± 0.03 (0.024)	267.6 ± 3.1 (0.547)	279.2 ± 3.2 (0.570)

Added time is 145.0–279.2 μs, or 0.444–0.570% of a complete decode step. Leaf state and merge dominate this cost; KV planning contributes 7.78–11.60 μs per step. Leaf state and merge remain necessary because shared KV delivery does not combine requests' private softmax states. Tile dispatch adds no replay-time GPU kernel or workspace under steady-state CUDA Graph execution. Extra workspace is 12.037 MiB per TP rank, mostly leaf state.

5 CONCLUSION

FLEXBI separates a fixed request-local numerical contract from its physical schedule to restore flexibility in batch-invariant LLM serving. FLEXTILE adapts GEMM output tiles while preserving each output's reduction trace. FLEXLOAD shares identical physical KV-page loads while preserving each query's attention reduction. Online serving across two models and two datasets shows 25–33% higher output throughput and 16–32% lower P99 time to first token than same-engine BI.

AI USE STATEMENT

We used generative AI tools to assist with writing and to polish the wording, grammar, and clarity of the manuscript. The authors take responsibility for the final text and all technical claims and results.

ETHICS STATEMENT

This work studies the efficiency and numerical reproducibility of LLM serving. Our trace-based evaluation uses publicly released, anonymized workload metadata and reconstructs token sequences from anonymous block identities rather than recovering users' original prompts. Batch invariance supports reproducible evaluation and auditing of model execution.

REPRODUCIBILITY STATEMENT

Section 3 specifies the numerical contracts and the physical schedules they admit, and Appendix B provides integration details and planning algorithms. Section 4 describes the hardware, models, and baselines. Appendices D.3 and D.2 document the workload construction, paired arrival protocol, metric aggregation, and measurement boundaries. Appendix C reports the request-level bitwise validation procedure and the compared logit and persistent-KV value counts.

REFERENCES

Marah Abdin, Jyoti Aneja, Hany Awadalla, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024. URL https://arxiv.org/abs/2404.14219.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in LLM-based multi-agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19888–19905. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.acl-long.912. URL https://aclanthology.org/2026.acl-long.912/.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/98ed250b203d1ac6b24bbcf263e3d4a7-Abstract-Conference.html.

Tri Dao, Daniel Y. 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. URL https://arxiv.org/abs/2205.14135.

Gemma Team, Morgane Riviere, Shreya Pathak, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024. URL https://arxiv.org/abs/2408.00118.

David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. doi: 10.1145/103162.103163.

Raja Gond, Aditya K. Kamath, Ramachandran Ramjee, and Ashish Panwar. LLM-42: Enabling determinism in LLM inference with verified speculation. *arXiv preprint arXiv:2601.17768*, 2026. URL https://arxiv.org/abs/2601.17768.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Horace He and Thinking Machines Lab. Defeating nondeterminism in LLM inference. *Thinking Machines Lab: Connectionism*, September 2025. doi: 10.64434/tml.20250910. URL https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023. URL https://arxiv.org/abs/2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024. URL https://arxiv.org/abs/2401.04088.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-throughput LLM inference with shared prefixes. *arXiv preprint arXiv:2402.05099*, 2024. URL https://arxiv.org/abs/2402.05099.

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 Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, et al. StarCoder 2 and The Stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024. URL https://arxiv.org/abs/2402.19173.

Joel D. Molina. Deterministic LLMs: A practical forensic framework for verifiable and reproducible local LLM inference. In *2026 14th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–6, 2026. doi: 10.1109/ISDFS69419.2026.11458980.

NVIDIA. CUDA programming guide: CUDA graphs. CUDA Programming Guide, 2026. URL https://docs.nvidia.com/cuda/cuda-programming-guide/04-special-topics/cuda-graphs.html. Accessed September 25, 2026.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. URL https://arxiv.org/abs/2508.10925.

Muhammad Osama, Duane Merrill, Cris Cecka, Michael Garland, and John D. Owens. Stream-K: Work-centric parallel decomposition for dense matrix-matrix multiplication on the GPU. *arXiv preprint arXiv:2301.03598*, 2023. URL https://arxiv.org/abs/2301.03598.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170. USENIX Association, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

SGLang Project. Deterministic inference. SGLang v0.5.16 documentation, 2026. URL https://github.com/sgl-project/sglang/blob/v0.5.16/docs_new/docs/advanced_features/deterministic_inference.mdx.

Team OLMo, Pete Walsh, Luca Soldaini, et al. 2 OLMo 2 furious. *arXiv preprint arXiv:2501.00656*, 2025. URL https://arxiv.org/abs/2501.00656.

vLLM Project. Batch invariance. vLLM v0.26.0 documentation, 2026. URL https://docs.vllm.ai/en/v0.26.0/features/batch_invariance/.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a. URL https://arxiv.org/abs/2505.09388.

An Yang, Baosong Yang, Beichen Zhang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025b. URL https://arxiv.org/abs/2412.15115.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/dbf02b21d77409a2db30e56866a8ab3a-Abstract-Conference.html.

Jinjun Yi, Zhixin Zhao, Yitao Hu, Ke Yan, Weiwei Sun, Hao Wang, Laiping Zhao, Yuhao Zhang, Wenxin Li, and Keqiu Li. PAT: Accelerating LLM decoding via prefix-aware attention with resource efficient multi-tile kernel. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1396–1412. ACM, 2026. doi: 10.1145/3779212.3790200. URL https://arxiv.org/abs/2511.22333.

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.

Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in LLM inference. In *Advances in Neural Information Processing Systems*, volume 38, pp. 169819–169851, 2025. doi: 10.52202/085713-5653.

Ziyang Zhang, Xinheng Ding, Jiayi Yuan, Rixin Liu, Huizi Mao, Jiarong Xing, and Zirui Liu. Deterministic inference across tensor parallel sizes that eliminates training–inference mismatch. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2511.17826.

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. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

A Extended related work

Deterministic and batch-invariant inference. Prior work identifies batch composition, reduction order, tensor parallelism, and hardware changes as sources of numerical variation in LLM inference (He & Thinking Machines Lab, 2025; Yuan et al., 2025). TBIK fixes reductions across tensor-parallel sizes and addresses training-to-inference mismatch (Zhang et al., 2026). LLM-42 speculatively generates candidate windows, periodically replays them under a fixed-shape verifier, and rolls back mismatches (Gond et al., 2026). Within a fixed execution platform, FlexBI asks which physical schedules remain available after a request's numerical computation has been fixed. Its contracts retain the reduction behavior that BI requires while exposing independent output placement and read-only data movement to runtime scheduling.

Shared-prefix Attention. PagedAttention and RadixAttention share KV storage and avoid repeated prefix computation (Kwon et al., 2023; Zheng et al., 2024). Hydragen reduces decode work by separating a shared prefix from request-specific suffixes and combining their Attention results (Juravsky et al., 2024). PAT groups queries with a shared prefix to reduce repeated KV loads (Yi et al., 2026). These methods establish the value of exploiting shared prefixes. FlexLoad targets the BI setting, where a sharing boundary that follows the current cache state can change the softmax reduction. It fixes logical leaf and merge boundaries, then shares the loads of leaves whose physical page sequences match exactly.

Dynamic LLM serving. Iteration-level scheduling in Orca, paged memory in vLLM, and radix caching in SGLang adapt execution to changing request populations (Yu et al., 2022; Kwon et al., 2023; Zheng et al., 2024). FlashInfer adapts attention load balancing to request dynamism while retaining CUDA Graph compatibility (Ye et al., 2025). Those systems create the runtime variation that motivates FlexBI: the useful row count and the physical page-sharing graph change throughout a workload. FlexBI asks how BI can coexist with the workload adaptation on which these serving systems rely. It integrates with their existing schedulers and cache managers rather than defining a new admission or batching policy.

B Implementation and algorithms

B.1 Framework integration

The adapters replace contracted operators at two phase-specific call sites while retaining each engine's scheduler and paged KV cache. The GEMM call site supplies a contract identifier and active matrix shape to FlexTile. The single-token decode attention call site supplies current sequence lengths, page tables, query metadata, and masks to FlexLoad. Multi-token prefill retains the engine's BI attention path and the same logical KV layout.

CUDA Graph replay reads dynamic plans from fixed-address buffers. Each captured shape bucket uses its admitted FlexTile schedule. Before replay, the engine refreshes sequence lengths and page tables. The GPU planner rewrites the task buffer and device-side count for FlexLoad. The cache manager retains every referenced page until the scheduler step completes.

B.2 Numerical contracts

The evaluated FlexTile contract fixes the arithmetic that determines each output. It uses BF16 inputs, FP32 accumulation, ascending 64-element K blocks, one reduction chain per output, an optional FP32 bias, and one output cast. One CTA writes each output element. An admitted shape without a calibrated FlexTile descriptor uses a same-contract fallback.

The evaluated FlexLoad contract fixes each query's attention reduction independently of the sharing plan. It uses the online-softmax recurrence (Dao et al., 2022), 512-token logical leaves, 128-token steps, FP32 partial state, and a fixed left-to-right leaf merge. Full-context and sliding-window masks enter the same per-query reduction graph and remain independent of the sharing plan. The evaluated FlexLoad decode path supports BF16 causal multi-head attention (MHA), grouped-query attention (GQA), and multi-query attention (MQA). It also supports paged KV, sliding-window masks, and decode CUDA Graphs.

B.3 CONTRACT-PRESERVING PLANNING

Algorithm 1 separates offline admission and calibration from runtime dispatch. Structural admission checks output coverage and the contracted reducer to establish trace equivalence for each admitted descriptor. Compiled bitwise probes check the generated kernels on tested inputs. Calibration selects a low-latency admitted schedule for each shape and coalesces adjacent M values with the same choice into intervals. Runtime dispatch uses the contract identifier, hardware target, numerical format, and active matrix shape to find the selected interval. The model name and tensor values do not enter this choice.

Algorithm 1 Contract-preserving admission and dispatch in FLEXTILE.

Require: Contract Γ; target h; format f; shapes $\mathcal{W}$; descriptors $\mathcal{D}$; bitwise probes $\mathcal{X}$
Ensure: Dispatch table T whose descriptors match Γ structurally and pass compiled probes

for all $w = (M, K, N) \in \mathcal{W}$ **do**
  $\mathcal{C}_w \leftarrow \emptyset$
  for all $d \in \mathcal{D}(w)$ **do**
    if d.CONTRACT $= \Gamma$ **and** COVERSONCE(d, w) **and** PRESERVESREDUCER(d, Γ, w) **then**
      $k \leftarrow$ COMPILE(Γ, d)
      if BITSAGREE$(k, \mathcal{X})$ across positions and schedules **then**
        $\mathcal{C}_w \leftarrow \mathcal{C}_w \cup \{d\}$
      end if
    end if
  end for
  if $\mathcal{C}_w \neq \emptyset$ **then**
    $T[\Gamma, h, f, w] \leftarrow \arg\min_{d \in \mathcal{C}_w}$ LATENCY(d, w)
  end if
end for
$T \leftarrow$ COALESCEMINTERVALS(T)
Runtime: $d \leftarrow T[\Gamma, h, f, (M, K, N)]$
if d is absent **then**
  $d \leftarrow$ SAMECONTRACTFALLBACK(Γ, h, f, M, K, N)
end if
LAUNCH(Γ, d)

Rows may share a KV load only when their complete ordered page vectors match within a logical leaf. Algorithm 2 uses the first-page key to narrow the comparison set and full page-vector equality to form sharing classes.

The task list is valid for one scheduler step. Task order, class size, and physical packing may change between steps. Each (q, r) retains private leaf state, and each query retains its logical merge order.

B.4 COMPLEXITY

Let $Q = |\mathcal{Q}|$ be the number of active query rows, $R = |\mathcal{R}|$ the number of logical leaves, and $V = |\mathcal{V}|$ the number of valid query–leaf pairs. Let p be the number of pages in one leaf, n_h the number of query heads per rank, and d the head dimension. For workload shapes $\mathcal{W}$ and candidate sets $\mathcal{D}(w)$, FLEXTILE admission examines $O(\sum_{w \in \mathcal{W}} |\mathcal{D}(w)|)$ descriptors offline. Runtime dispatch looks up a precomputed M interval with $O(1)$ step-local metadata.

FLEXLOAD scans Q candidates for each valid query–leaf pair and compares complete page vectors only when their first pages match. Let $\bar{c}$ be the average number of first-page matches per valid pair in the workload. Average plan construction takes $O(V(Q + \bar{c}p) + Vp)$ time, including packing and task emission. The ordered merge visits each private partial state once and takes $O(Vn_hd)$ time. The plan metadata and partial states use $O(Vp + Vn_hd)$ space.

Algorithm 2 Exact-page planning and state-isolated execution in FLEXLOAD.

Require: Active query rows $\mathcal{Q}$; valid query–leaf pairs $\mathcal{V} \subseteq \mathcal{Q} \times \mathcal{R}$; page table P; logical leaves $\mathcal{R}$; row capacity C
Ensure: Tasks $\mathcal{T}$ that cover every $(q, r) \in \mathcal{V}$ once and group only identical complete page vectors

$\mathcal{T} \leftarrow \emptyset$
for all $(q, r) \in \mathcal{V}$ **do**
  $z_{q,r} \leftarrow$ INITPRIVATELEAFSTATE(q, r)
end for
for all $r \in \mathcal{R}$ **do**
  $\mathcal{Q}_r \leftarrow \{q \in \mathcal{Q} : (q, r) \in \mathcal{V}\}$
  $\mathcal{B} \leftarrow$ GROUPBYFIRSTPAGE$(\mathcal{Q}_r, P, r)$
  for all $B \in \mathcal{B}$ **do**
    $\mathcal{E} \leftarrow B/\sim_r$, where $q_a \sim_r q_b \Leftrightarrow P(q_a, r) = P(q_b, r)$
    for all $E \in \mathcal{E}$ **do**
      for all $G \in$ PACKROWS(E, C) **do**
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{(r, P(q, r), G)\}$ for any $q \in G$
      end for
    end for
  end for
end for
for all $(r, p, G) \in \mathcal{T}$ in any physical order **do**
  for all KV fragment $x \in p$ in contracted token order **do**
    Load x once
    for all $q \in G$ **do**
      ADVANCEPRIVATELEAFSTATE$(z_{q,r}, q, x)$
    end for
  end for
end for
for all $q \in \mathcal{Q}$ **do**
  MERGELEAVES$((z_{q,r})_{r:(q,r)\in\mathcal{V}})$ in contracted logical order and write q's output
end for

C REQUEST-LEVEL BITWISE VALIDATION

The request-level test checks whether a target's values remain bitwise identical as its execution context changes. We run a seeded stress test for both models and engines on A100 TP2 in BF16, with native BI and FLEXBI checked separately. Each configuration uses four target prompts of different lengths, 86–430 tokens for Qwen3 and 87–430 for StarCoder2. Each target runs alone and in 15 other contexts: seven directed cases and eight pseudorandom cases (seed 20260925 plus target index). The contexts vary batch size (1–4), target position, peer departures during generation, prefix-cache state, and physical KV placement. Each target generates 32 tokens. Shorter peers leave its batch during decoding.

The captured traces confirm cache, sharing, and placement variation. Across the contexts of each configuration, they contain 0, 16, 32, and 64 cached target-prefix tokens and 0–3 peers sharing the target's first physical KV page. Every target also has at least 11 distinct page mappings. Both engines capture decode CUDA Graph sizes 1–4. vLLM also captures mixed prefill–decode graphs. We compare all 32 generated token IDs and full-vocabulary raw logits before token selection. We also read the corresponding persistent-KV rows from the physical cache and align them by logical position across all layers and TP ranks. Each candidate is compared bitwise with the target-alone reference in the same configuration.

An eager run adds decoder-block coverage that Python hooks cannot capture during CUDA Graph replay. The eager run uses the same targets, contexts, and 32 decode steps. At every step we compare all 40 block outputs on both TP ranks and the final hidden vector entering the LM head. Qwen3 block outputs contain both hidden and residual tensors. StarCoder2 block outputs contain one tensor. The final hidden vector is captured on both ranks in vLLM and on rank 0 in SGLang. All 5,033,164,800 compared block-output values match bitwise across the four FLEXBI configurations.

Table 2 separates the CUDA Graph comparisons from the eager final-hidden comparison. Final hidden comes from the eager run. Tokens, logits, and KV come from the CUDA Graph run. Each n/e entry gives compared values and bitwise mismatches for FLEXBI. Native BI has the same denominators and zero mismatches for the CUDA Graph columns.

Table 2: Request-level FLEXBI checks over four targets, 60 compared contexts, and 32 decode steps each. n/e gives compared values and bitwise mismatches.

Model	Engine	Tokens n/e	Output logits n/e	Final hidden n/e	KV n/e
Qwen3-14B	vLLM	1,920/0	291,717,120/0	19,660,800/0	157,286,400/0
Qwen3-14B	SGLang	1,920/0	291,717,120/0	9,830,400/0	157,286,400/0
StarCoder2-15B	vLLM	1,920/0	94,371,840/0	23,592,960/0	78,643,200/0
StarCoder2-15B	SGLang	1,920/0	94,371,840/0	11,796,480/0	78,643,200/0

The eager native-BI Qwen3/vLLM control has bitwise mismatches in hidden states despite matching all 1,920 token IDs. Specifically, 5,101,092 of 19,660,800 final-hidden values and 356,667,938 of 1,572,864,000 block-output values differ. We therefore report eager hidden and block outputs for FLEXBI alone. The native-BI zero-mismatch control in Table 2 applies to the CUDA Graph logits and KV experiment.

The test does not exercise staggered arrivals during the target's decoding or a sliding-window rollover. The artifacts retain native-BI controls, per-case tensor hashes, logical-to-physical KV locations, per-rank FLEXTILE dispatch, measured coverage, and commands.

D EXPERIMENTAL SETUP AND PROTOCOLS

D.1 REPRODUCIBILITY SETTINGS

The experiments use two NVIDIA A100-SXM4-80GB GPUs with driver 580.159.03, TP2, and BF16. Both engine environments use Python 3.12.13, PyTorch 2.11.0+cu128, CUDA Toolkit 12.8, Triton 3.6.0, FlashInfer 0.6.14, and NCCL 2.28.9. All systems use identical model weights and the same per-model context limit. The artifacts retain the framework revisions, model and dataset hashes, launch configurations, graph buckets, KV budgets, continuation hashes, package freezes, and raw client timings.

D.2 DATASET CONSTRUCTION AND TRACE SLICES

The workload builder reads the first 30 minutes of the ToolAgent and Conversation traces released with Mooncake FAST'25 (Qin et al., 2025). It retains records with a positive source output length whose input and output fit the common 16,384-token context limit. This construction filter fixes source rows across models. Table 3 lists each model's configured runtime context limit. The builder sorts these records by timestamp and source row, then selects the first 4,000 without ranking for prefix reuse or performance. We freeze three non-overlapping slices from each pool and reuse their source rows across models and systems. Because the public traces contain anonymous block identities rather than prompt text, the builder deterministically reconstructs model-valid token IDs while preserving complete prefix relations.

Table 3: Models used in the evaluation (Yang et al., 2025a; Lozhkov et al., 2024). Context is the configured evaluation limit.

Model	Layers	Hidden size	Attention	Context
Qwen3-14B	40	5,120	Full	32,768
StarCoder2-15B-Instruct-v0.1	40	6,144	Sliding (4,096)	16,384

Table 4 compares each prepared pool with the three measured slices. The statistics exclude compilation warmup requests.

Table 4: Prepared workload pools and measured trace slices. Columns report request counts, token-length percentiles, and prefix potential.

Workload	Scope	Requests	Input P50/P95	Output P50/P95	Prefix potential
ToolAgent	Full	4,000	6,176/11,724	26/561	66.9%
ToolAgent	Trace 1	240	6,190/12,940	26/575	54.3%
ToolAgent	Trace 2	240	6,236/12,444	27/568	56.3%
ToolAgent	Trace 3	240	6,217/11,122	24/467	56.6%
Conversation	Full	4,000	5,001/14,057	346/683	33.7%
Conversation	Trace 1	120	5,523/14,358	359/622	11.4%
Conversation	Trace 2	120	6,188/14,350	347/655	8.9%
Conversation	Trace 3	120	5,700/15,199	341/600	10.4%

Prefix potential is the fraction of input tokens covered by complete 512-token prefix blocks whose cumulative block hash appeared earlier in the same scope. The full pool and each measured slice reset their prefix history independently. The shorter slices have less prior history in which a cumulative block hash can appear. This statistic assumes unlimited retention and is neither an observed cache-hit rate nor a same-step KV-sharing rate. The online artifacts report realized cache hits, concurrent consumers, and evictions separately.

D.3 Online serving protocol and metrics

Paired trace arrivals. The online experiment preserves each trace slice's request order and attributes. A frozen Poisson arrival stream supplies the request times. For each workload, a recorded seed generates unit-rate exponential gaps, which the runner accumulates and scales to the target offered rate. The cohort contains at least 32 requests and covers five minutes at the model–workload anchor rate. Each run uses a disjoint segment of the frozen arrival stream, and the client submits requests without waiting for earlier requests to complete.

Pairing holds the offered work and initial service state fixed across systems. For a fixed model, workload, rate, and session, every system receives the same requests, arrival times, output lengths, and continuation. The three sessions use non-overlapping trace slices and arrival segments. Each measured run starts from the same empty prefix-cache state and retains the same graph configuration and KV budget. The artifact records the seeds, offsets, request identifiers, and workload hashes.

Online metrics. For each request, TTFT is the elapsed time from its scheduled arrival until the client receives its first output token:

$$\mathrm{TTFT}_i = t_i^{\text{first token}} - t_i^{\text{scheduled arrival}}.$$

TTFT includes admission delay, queueing, prefill, and first-token transmission. End-to-end latency runs from scheduled arrival to completion of the streamed response. For a request that returns $D_i > 1$ tokens, request-level time per output token (TPOT) is the time between its first and last token divided by $D_i - 1$. The client records completion after the response stream ends. Single-token responses have no TPOT.

Output TPS measures completed output over the full client makespan.

$$\text{output TPS} = \frac{\text{output tokens from completed requests}}{\text{last completion time} - \text{first scheduled arrival}}.$$

The makespan includes request injection, execution, and final queue drain.

Run-level latency percentiles use the sorted request values. To compute P50 and P99 for TTFT, E2E, and request-level TPOT, we sort the request-level values within one run and evaluate quantiles at positions $0.50(n-1)$ and $0.99(n-1)$. We linearly interpolate when a position falls between observations.

Aggregation across runs. Each plotted point is the arithmetic mean of three run-level measurements. We compute each latency percentile within a run before averaging the three values. Requests from separate runs remain separate for percentile calculation.

Completion. The artifacts retain errors, timeouts, and completion fraction. Reported tail latency describes the completed finite cohorts, including final queue drain, rather than a steady-state service-level objective (SLO).

Controlled complete-call timing. RQ2 includes uncached suffix prefill, decode, planning, merge, logits, and output completion after identical cache priming.

Operator-level ablation. RQ3 runs on A100 TP2 in BF16 and compares each FlexBI mechanism with the corresponding native BI operator. For FLEXTILE, Figure 5 uses $M = 64$, while Figure 8 repeats the same grid at $M \in \{8, 16, 32, 128\}$. The grid is the cross-product of TP2-local K and N dimensions from the QKV and dense FFN shapes of ten model configurations across eight model families. The dense-model shapes come from Qwen3, Qwen2.5, Llama 3.1, Mistral 7B, OLMo 2, Phi-3 Mini, and Gemma 2 (Yang et al., 2025a;b; Grattafiori et al., 2024; Jiang et al., 2023; Team OLMo et al., 2025; Abdin et al., 2024; Gemma Team et al., 2024). The MoE-model shapes come from Mixtral, Phi-3.5-MoE, and gpt-oss (Jiang et al., 2024; Abdin et al., 2024; OpenAI, 2025). Each cell times the admitted schedule and records its selected $\mathrm{BM} \times \mathrm{BN}$ tile.

The FLEXLOAD operator test isolates the effect of shared KV loads at a fixed batch size. For FLEXLOAD, $B = 32$ contains one group of eight requests that share physical KV pages and 24 independent requests. The six TP2-local attention configurations vary the query-head count, KV-head count, and head dimension. Native BI attention with one split is the baseline. Every GEMM output, attention output, and per-query attention state is compared bitwise with its corresponding native BI operator.

Runtime-overhead measurement. RQ4 uses Qwen3-14B in vLLM on A100 TP2 with BF16, $B = 32$, and 2,048 generated tokens per request. The independent-prefix case uses $I = 4224$, $H = 4096$, and $L = 0$. The shared-prefix case uses $I = 16512$ and $H = L = 16384$, with eight of 32 requests sharing the physical prefix and the other 24 using independent prefixes of the same length.

The timing denominator amortizes complete-call time, while added time covers planning and merge. Runtime values are the mean and sample standard deviation of three session-level measurements. We report amortized complete-call time per decode step as the complete-call makespan divided by 2,048 decode steps. Added time includes KV planning once per step and leaf merge in each of 40 layers. The added-time accounting excludes useful leaf attention computation.

Additional workspace counts the compact plan and FP32 leaf-state buffers introduced by the method. Existing block tables, sequence lengths, the KV cache, and output buffers remain outside this accounting.

E ADDITIONAL RESULTS

E.1 LARGE-MODEL EVALUATION ON H100

The H100 experiment tests whether same-engine throughput gains extend to two 70B-class dense models on H100 TP4 (Figure 6). We compare vLLM-BI with FlexBI-vLLM in vLLM v0.26.0 on Llama-3.1-70B-Instruct and Qwen2.5-72B-Instruct (Grattafiori et al., 2024; Yang et al., 2025b). Each paired run uses the same four H100 80GB GPUs with TP4, BF16, FlashAttention 2 (Dao, 2024), prefix caching, and the default tensor-parallel communication path. The server sets `max_num_seqs=32` and `max_num_batched_tokens=4096`. CUDA Graph capture uses `FULL_AND_PIECEWISE` for batch sizes 1, 2, 4, 8, 16, and 32.

Both systems receive paired Mooncake ToolAgent and Conversation traces reconstructed with each model's tokenizer. Llama uses offered rates of 0.05, 0.1, 0.2, and 0.4 requests/s, with 32 measured requests per run. Qwen uses 0.1, 0.2, 0.4, and 0.8 requests/s, with 60 measured requests per run. The three sessions draw non-overlapping trace slices.

The paired H100 comparison contains 96 runs and 4,416 requests. Each plotted point averages three session-level measurements. Runtime dispatch records show M-dependent GEMM schedules on

every TP rank. The request-level bitwise comparisons in Table 2 cover A100 TP2. The H100 runs provide separate serving-performance evidence.

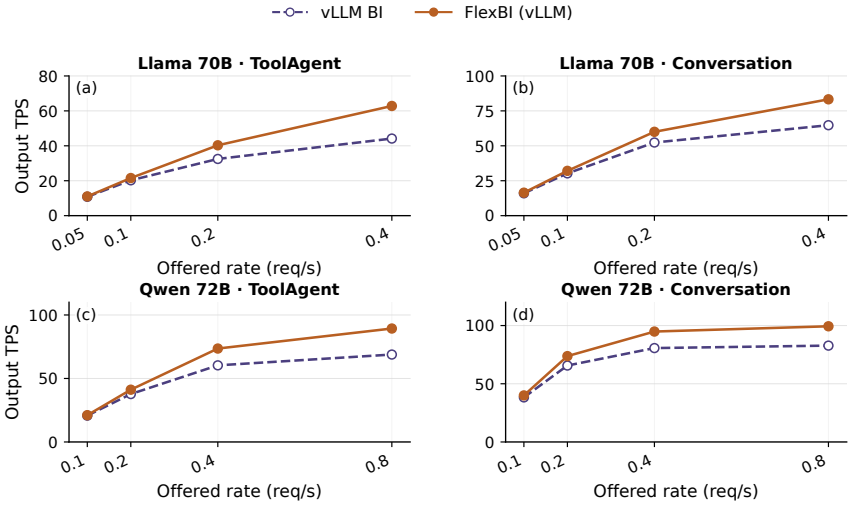


Figure 6: Output throughput of vLLM-BI and FlexBI-vLLM across offered request rates.

The throughput advantage is largest at the upper measured offered rates. At the highest measured Llama rate, FlexBI-vLLM reaches 62.8 versus 44.2 output tokens/s on ToolAgent and 83.3 versus 64.7 on Conversation. These differences are 42.3% and 28.6%, respectively. At the highest measured Qwen rate, FlexBI-vLLM reaches 89.3 versus 68.8 output tokens/s on ToolAgent and 99.4 versus 82.8 on Conversation, gains of 29.8% and 20.1%. Across all 16 paired points, the gains range from 1.7% to 42.3%.

The upper-rate throughput values describe completed finite cohorts.

E.2 Additional A100 online metrics

The additional A100 latency metrics favor FlexBI throughout the measured grid (Figure 7). Relative to same-engine BI, end-to-end latency improves at P50 and P99, as do P50 TTFT and request-level TPOT. The figure covers every online model, workload, offered rate, and system from Figure 3. All 43,200 offered requests completed without error across the two formal campaigns. Appendix D.3 defines the metrics and three-run aggregation.

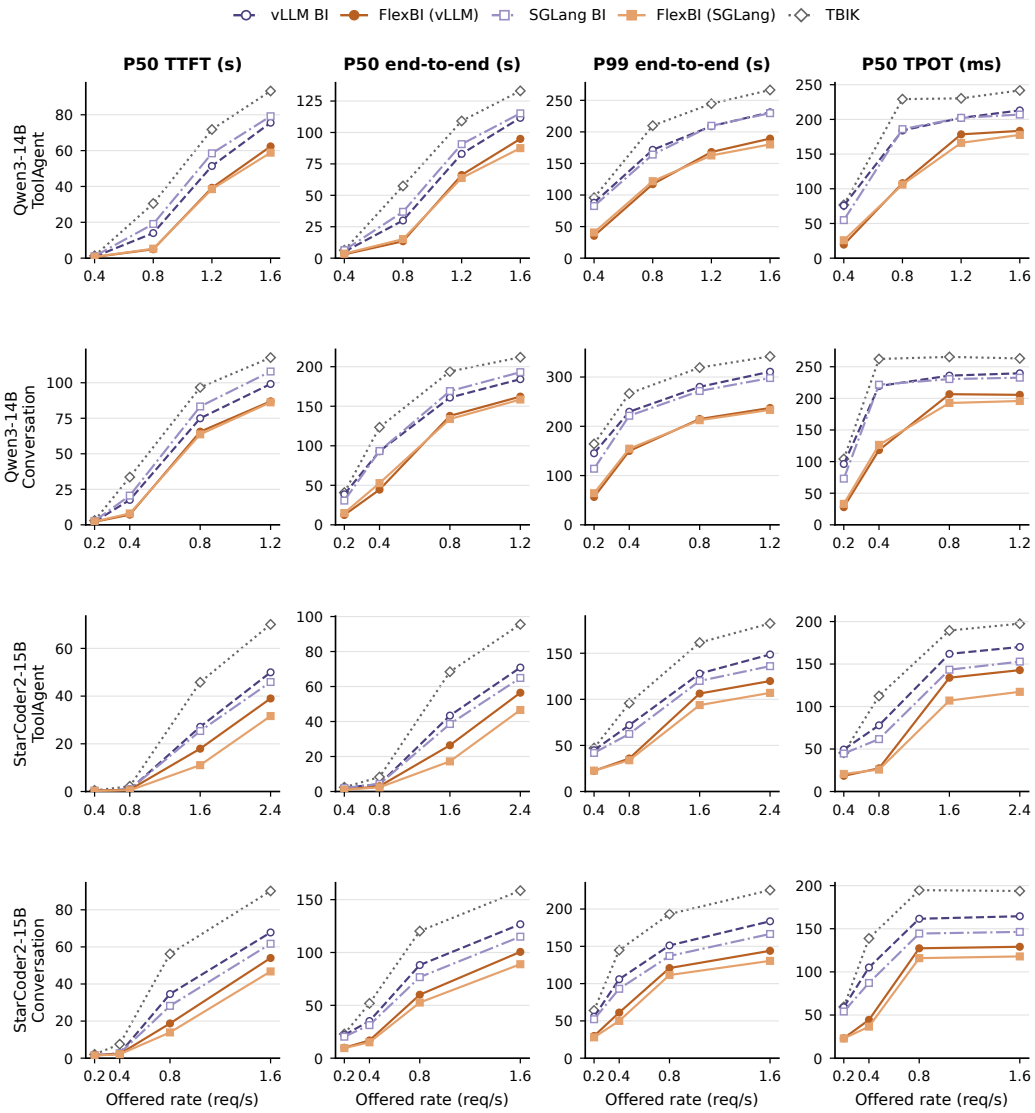


Figure 7: Additional A100 TP2 online latency metrics.

E.3 Additional FlexTile input-row counts

Figure 8 repeats the $M = 64$ operator grid in Figure 5 at $M \in \{8, 16, 32, 128\}$. The same contracted reducer and native BI baseline apply at every input-row count.

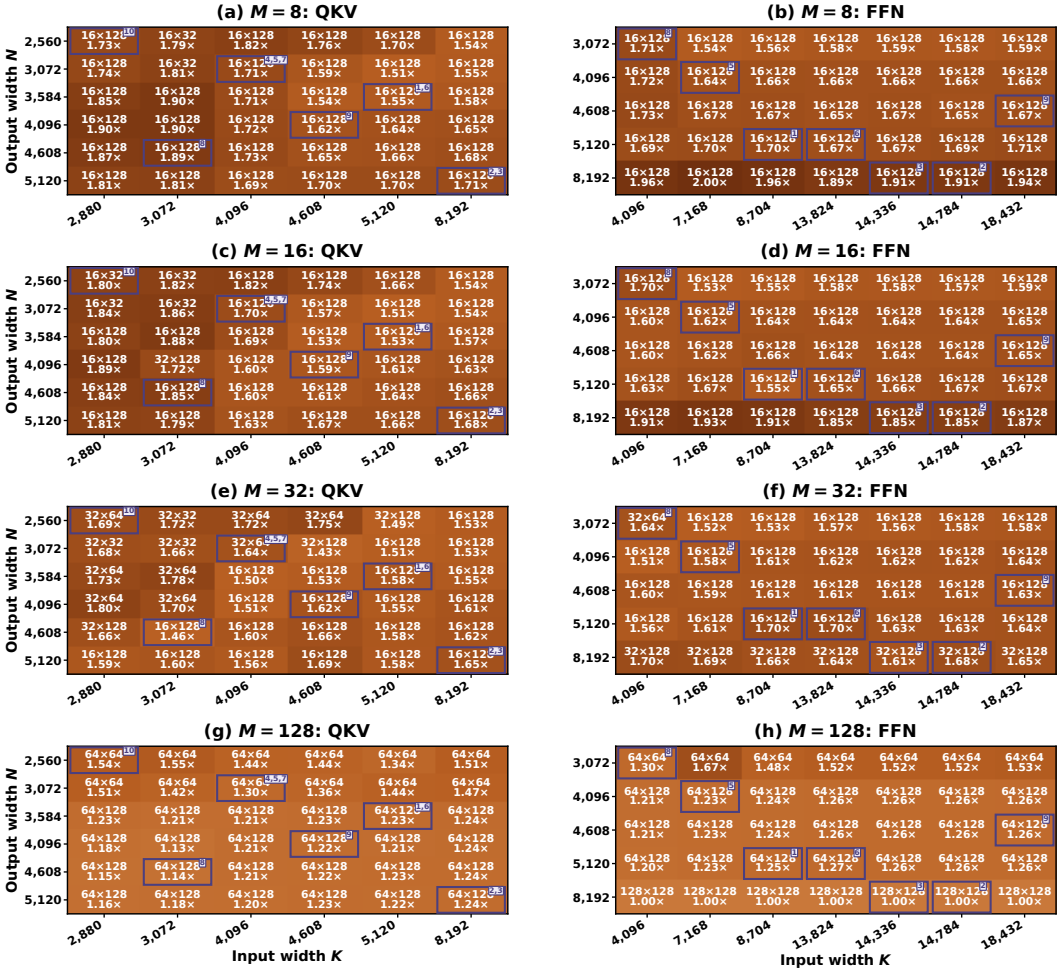


Figure 8: FlexTile at additional input-row counts. Cells use the encoding and model key from Figure 5.