FoldAttention: Declared-Reference Softmax for Fast Decode and Deterministic Backward

Anonymous authors
Paper under double-blind review

Abstract

Autoregressive decode repeatedly streams a growing KV cache, making attention a major cost at long context. Existing high-performance kernels use online softmax, which discovers a row's normalization reference as it scans keys. Earlier contributions therefore remain provisional and may require rescaling. We argue that the reference need not be discovered: softmax is invariant to a common shift, so the reference only has to keep the weights in range. We present FoldAttention, an additive formulation of softmax attention that fixes a finite reference Z_i before scanning the KV cache. Each weight $2^{s_{ij}-Z_i}$ is then final when computed, so contributions add across disjoint key ranges and their quotient equals softmax attention in real arithmetic. We use this property to develop two techniques for Hopper decode: (1) final weights gate key and value reads before the bytes are fetched, and a per-call depth T cuts keys below 2^{-T} while keeping their mass, and (2) additive partials compose split KV and shared-prefix cascades without rescaling. On H100 at $T = 16$, FoldAttention decodes seven real-model generations 1.36–2.30$\times$ faster than the fastest BF16 baseline, and up to 3.09$\times$ faster across MHA and GQA shapes, at an error within 1.5% of the lowest BF16 error on six of the seven; reading every key, it is 1.14–1.30$\times$ faster at matched error. We validate on Qwen3-8B that a whole decode step is up to 1.46$\times$ faster while likelihood and long-context accuracy match those under BF16 kernels. The same principle makes the backward deterministic: CTAs round bounded partial gradients onto an integer grid declared before the reduction and add them in any order. FoldAttention thereby removes the determinism tax: its deterministic backward is up to 1.84$\times$ faster than deterministic FlashAttention-3/4 and 1.05$\times$ faster than the fastest nondeterministic kernel.

1 Introduction

Autoregressive decode reads the KV cache to produce one token per request. It is therefore bound by memory bandwidth, and its cost grows with context and batch size. At long context, attention alone takes up to half of a decode step (Yi et al., 2026). Training has a separate bottleneck in the attention backward. Many thread blocks (CTAs) contribute to each gradient, and their arrival order determines the result's bits. Reproducing a training run requires those bits to remain fixed (Qiang et al., 2026; He & Thinking Machines Lab, 2025).

High-performance kernels, FlashAttention and its successors among them, use online softmax (Milakov & Gimelshein, 2018; Dao et al., 2022; Dao, 2023; Shah et al., 2024; Zadouri et al., 2026). Each key tile is weighted relative to a running maximum, and a later increase rescales the accumulated state. No weight is final until the last key is scored. Measured against the running maximum, which each cache split restarts, a weight only bounds its final value, so reads can be skipped only conservatively and split partials merge only by comparing maxima and rescaling. Backward reductions have an analogous dependency: FP32 addition rounds relative to the running total, so the arrival order of CTAs determines the bits. Deterministic modes pay a *determinism tax* to impose an order: up to 38% of throughput in FlashAttention-3 and 25% in FlashAttention-4 (Qiang et al., 2026; Zadouri et al., 2026).

Both dependencies arise because a numerical scale is discovered during parallel work. FoldAttention instead declares the scale before that work begins. For decode, softmax invariance means that the

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.

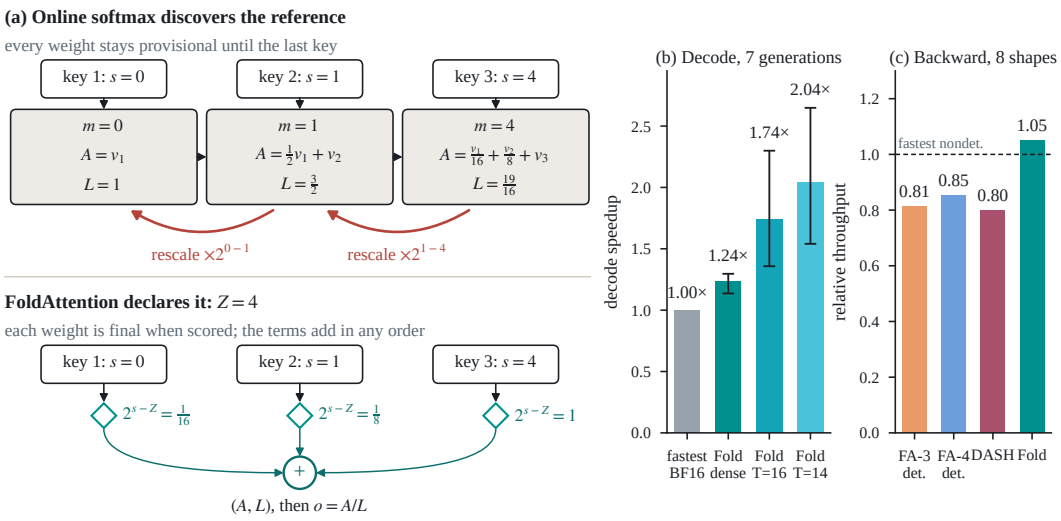

Figure 1: *(a)* Online softmax rescales A and L each time the running maximum rises; with $Z = 4$ fixed first, each weight is final and the terms add in any order. *(b)* Decode speedup over the fastest BF16 kernel on seven generations (geometric mean and range). *(c)* Deterministic backward throughput relative to the fastest nondeterministic kernel, over eight shapes.

reference only has to keep weights inside the exponent range of FP32 accumulation and BF16 tensor-core operands. A cheap per-row estimate lands within 29 binades of the true log-sum-exp on every row we measured, up to 128K keys. Prior work fixes or freezes a reference to reduce rescaling (Hong et al., 2024; Zadouri et al., 2026; Sun et al., 2026), or drops blocks below a pseudo-maximum together with their mass (Liu et al., 2026). FoldAttention uses a declared reference more broadly: splits share it, final weights gate reads, and cut keys still contribute to the denominator. For backward, every CTA derives the same declared rounding grid, so partial gradients add as integers in any order.

We present FoldAttention, which fixes a finite reference Z_i for each query row before any key is scanned (Figure 1a). We contribute:

1. **One additive contract** (Section 3) for split-KV and cascades, exact in real arithmetic with finite-precision range bounds, and an order-free integer reduction for backward.

2. **A decode kernel that reads by final weight** (Section 4). Keys are two INT8 planes with a per-key scale; the first plane gives each key's final weight, which decides whether the second plane and the value row are read. A depth T, set per call on the same cache, cuts keys below 2^{-T} while keeping their mass, a dial from BF16-kernel accuracy to fewer bytes. Shared prefixes are read once for every request that holds them.

3. **A deterministic backward at nondeterministic speed** (Section 5). CTAs round their partials onto a common power-of-two grid, folded for dQ into the softmax's own exponential, and add integers in any order.

Against tuned FlashAttention-3/4, FlashInfer, TensorRT-LLM XQA (NVIDIA Corporation, 2026c), cuDNN (NVIDIA Corporation, 2026a), and DASH (Qiang et al., 2026) on an H100, scored against one FP32 reference, FoldAttention at depth 16 decodes seven multi-step generations of Qwen3-30B-A3B (Yang et al., 2025), gpt-oss-20b (OpenAI, 2025), and GLM-4-9B (Team GLM, 2024) 1.36–2.30× faster than the fastest BF16 kernel. Its output error is within 1.5% of the lowest BF16 error in six cases, and it reaches 3.09× on a 1K–32K sweep of MHA and GQA shapes. On Qwen3-8B, a whole decode step is 1.13–1.46× faster, and likelihood, retrieval, and LongBench accuracy match those under BF16 kernels from 8K to 128K context. The deterministic backward is up to 1.84× faster than deterministic FlashAttention-3/4 and 1.05× faster than the fastest nondeterministic kernel, with bit-identical gradients across runs, batches, and packings.

2 Background

Online softmax and its merge. Let q_i be a query row, k_j, v_j the keys and values, and $s_{ij} = \log_2(e)\, q_i^\top k_j/\sqrt{D}$ the score in base 2, so attention (Vaswani et al., 2017) returns $o_i = \sum_j 2^{s_{ij}} v_j \big/ \sum_j 2^{s_{ij}}$. Online softmax (Milakov & Gimelshein, 2018) keeps a running maximum m, numerator A, and denominator L, and when a tile raises m to m' multiplies A and L by $2^{m-m'}$ before adding the tile's terms $2^{s_{ij}-m'}$. Split-KV decode (Dao et al., 2023) returns a partial state (o, ℓ) per split, with $\ell = \log L + m$, and merges two as

$$(o, \ell) \oplus (o', \ell') = \Big(\tfrac{e^{\ell} o + e^{\ell'} o'}{e^{\ell} + e^{\ell'}},\ \log\big(e^{\ell} + e^{\ell'}\big) \Big), \tag{1}$$

as FlashInfer, LeanAttention, and Hydragen do for split-KV, cascades, and shared prefixes (Ye et al., 2025; Sanovar et al., 2024; Juravsky et al., 2024). The merge is evaluated stably only by comparing ℓ and ℓ' and rescaling one side, the same step the scan takes.

Decode traffic and gradient reduction. A BF16 decode step reads 512 bytes of key and value per cached token at $D = 128$. The fastest BF16 kernel already reaches 93–99% of the H100's measured bandwidth at 16K keys and beyond, so a faster decode must read fewer bytes. FP8 caches (Micikevicius et al., 2022) halve this traffic but raise the output's FP32-relative error by 27–133$\times$ in our benchmarks. In backward, with $D_i = dO_i^\top o_i$ and $dS = P \circ (dO\, V^\top - D)$, each CTA holds a key block and adds a partial into every row of $dQ = dS\, K$ it touches. FP32 addition is not associative, so with atomic adds the bits of dQ follow the CTAs' finishing order, and deterministic modes serialize the adds (Qiang et al., 2026; Zadouri et al., 2026).

3 The additive contract

Fixed-reference softmax. Fix a finite reference Z_i for query row i before any key is read. For a set of keys I define

$$F_i(I) = (A_i^I, L_i^I) = \Big(\sum_{j \in I} 2^{s_{ij} - Z_i} v_j,\ \sum_{j \in I} 2^{s_{ij} - Z_i} \Big), \qquad o_i = A_i^I / L_i^I. \tag{2}$$

Every term depends on one score and on Z_i, and on nothing seen before or after it.

Proposition 1 (Exact additive softmax)**.** *Let Z_i be finite.* (i) *For disjoint key sets, $F_i(I) + F_i(J) = F_i(I \cup J)$.* (ii) *$A_i/L_i$ equals the softmax attention output of row i over the represented scores and values, for every Z_i.* (iii) *Let the weights be evaluated and accumulated in FP32 with flush to zero, over $n \le 2^{22}$ keys with $m_i = \max_j s_{ij}$ and $\nu = \max(1, \max_{j,d} |v_{jd}|)$. If $m_i - Z_i + \log_2(n\nu) < 127$, no weight, numerator, or denominator overflows, and flushed weights change L_i by a relative amount of at most $n\, 2^{-126-(m_i - Z_i)}$.*

Appendix A gives the proof. Where online softmax merges partial states with Equation 1, these pairs merge with $+$: no maximum is compared and no finished term is rescaled, so split-KV, shared-prefix cascades, and warp completion order are all groupings of one sum. The decode kernel applies the identity to inputs its gates choose (Section 4), and Section 6 measures its error against FP32.

Choosing the reference. Part (iii) is the only condition Z_i must meet. The BF16 operand that carries each weight into the tensor core has FP32's exponent range, so the condition is the same for the product. Z_i need not be the maximum or a bound on it. An estimate too high or too low scales every weight by the same factor, which cancels in A/L, so only range matters: with $n \le 2^{22}$ keys and values below 2^{16}, any Z_i within about 80 binades of the row's largest score keeps every weight and sum in range and loses less than 2^{-24} of L_i to flushing.

FoldAttention estimates Z_i per row and step as a log-sum-exp from 192 keys scored with the kernel's own coarse logits: the attention sink (Xiao et al., 2024) and the 63 most recent keys exactly, plus a trimmed stratified estimate from 128 stratum centers over the rest, with no state carried between steps. The true log-sum-exp sits -0.4 to 17.2 binades above Z_i over every row of the seven generations of Section 6, and -0.9 to 28.2 in Qwen3-8B at 64K–128K context: more than an FP16 weight holds, which is why a fixed FP16 reference needs a fallback (Hong et al., 2024), and far inside BF16's range

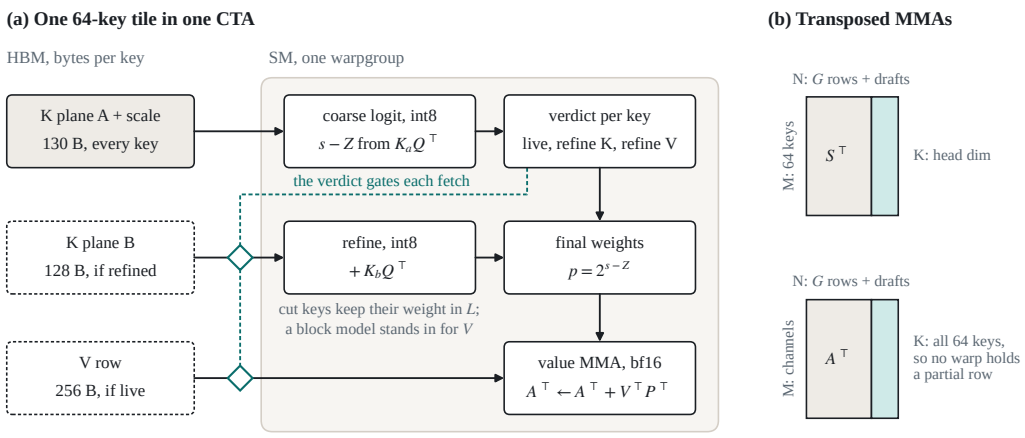

Figure 2: *(a)* One CTA on a 64-key tile, bytes per key at $D = 128$: plane A and the key's scale give its score against Z_i, and the verdict, ORed over the GQA group, gates the reads of plane B and the value row. *(b)* Keys or channels sit on M and the group's query rows (and any draft rows) on N, so the value product reduces over a whole tile.

(Appendix D). A cache can also certify the range after each step: a row whose $L_i = 2^{\ell_i - Z_i}$, with ℓ_i its base-2 log-sum-exp, leaves $[2^{-1}, 2^{100}]$, or whose output is not finite, is decoded again with $Z_i = \ell_i$. The check syncs with the host, so timed runs omit it; across 1.8 billion rows decoded in Qwen3-8B it would rerun 118, each with Z_i less than 1.6 binades above ℓ_i.

Integer-grid gradients. The backward needs the same declaration one level down: a rounding scale fixed before the parallel reduction, so arrival order cannot change the sum. It adds partials g_c from C CTAs into one gradient element. Given a bound $\sum_c |g_c| \le B$, choose the power of two $\alpha = 2^{b - \lceil \log_2 B \rceil}$ and sum integers:

$$\hat{g}_c = \mathrm{RNE}(\alpha g_c), \qquad g = \alpha^{-1} \textstyle\sum_c \hat{g}_c. \tag{3}$$

Proposition 2 (Order-free gradient sums)**.** *If $C < 2^{b+1}$, the $\hat{g}_c$ summed in $(b+2)$-bit two's-complement arithmetic give the same g in every order and grouping, no partial sum overflows, and $|g - \sum_c g_c| \le C\, B\, 2^{-b}$.*

This is pre-rounding in the sense of reproducible summation (Demmel & Nguyen, 2013; Ahrens et al., 2020); what attention adds is cheap bounds. With P a row-stochastic matrix, $\|v_j\| \le C_D \max |V|$ and Cauchy–Schwarz, the terms of one dQ element satisfy

$$\textstyle\sum_j |dS_{ij} K_{jd}| \le \left(\max_i \|dO_i\|\, C_D \max |V| + \max_i |D_i|\right) \max |K|, \tag{4}$$

where C_D is $\sqrt{D}$ rounded up to a power of two and the maxima run over one request's rows and one KV head. Every CTA's partial is a subset of these terms, so the right side bounds $\sum_c |g_c|$. The kernel widens it by 2^{-6} for the BF16 rounding of P and dS. A preprocess reduces the maxima with order-free integer maxima over their bit patterns, and every CTA evaluates the bound with the same instructions, so all arrive at the same grid: like Z_i, it is declared before the reduction, not discovered by it.

4 Decode kernel

FoldAttention uses a paged cache (Kwon et al., 2023), a per-step front kernel that writes the new token and estimates Z_i, and a split-KV kernel. The fixed reference governs every read decision in the split-KV kernel. Figure 2 shows one CTA's work on one tile.

Cache format. Keys and queries are rotated by an orthonormal Hadamard matrix, which keeps every $q^\top k$ and spreads outlier channels (Ashkboos et al., 2024). Each key is stored as two INT8

planes, $k_j = e_j(a_j + b_j/256)$, with a BF16 scale e_j per key rounded up so plane A never saturates. A key's code therefore depends on that key alone: no calibration pass, no headroom, and appended tokens are quantized as precisely as the prompt. The query is split the same way each step. Values are BF16, or two E4M3 (Micikevicius et al., 2022) planes under a power-of-two scale per layer. The kernel widens the E4M3 values to BF16 exactly by bit placement. Plane A and its scale cost 130 bytes per key at $D = 128$.

Weight-gated reads. Each step, a front kernel quantizes the query, appends the new token, and estimates every row's Z_i (Section 3). Each decode CTA is one warpgroup and walks its split in tiles of 64 keys; a request's splits interleave by tile, because live keys cluster in the most recent tiles and contiguous splits would leave them all to one CTA. An INT8 WGMMA scores every key against every row of the GQA (Ainslie et al., 2023) group, keys on M and rows on N (Figure 2b). Against the fixed Z_i, each score yields three final verdict bits: *live* ($s \geq Z_i - T$), *refine K*, and *refine V*, set when a weight is large enough for the second plane to matter at the error target; the refine thresholds follow each request's own length, so its gates do not depend on its batch. The bits are ORed over the group, and only then are bytes requested: plane B for refined keys, added through a second INT8 product, and value rows for live keys. A tile with no live key skips its value work. Against a running maximum these reads could be skipped only conservatively (Section 6.2). At $D = 64$, groups of up to four rows walk 128-key tiles that hold two keys in each row of the product (Appendix D.4).

Value product. Weights $p = 2^{s-Z_i}$ are formed in FP32 and fed to a BF16 WGMMA, $A^\top \leftarrow A^\top + V^\top P^\top$, with channels on M, rows on N, and the tile's 64 keys on the reduction dimension, so no warp holds a partial row. For dense decode and depths of 15 or more, the BF16 rounding residual of p rides a second product; L sums the weights the products use. Each split writes an FP32 pair (A, L), and a combine adds the pairs in a fixed slot order (Algorithm 1). Speculative drafts (Leviathan et al., 2023) and verification trees (Miao et al., 2024) are extra query columns masked to their ancestors. Because pairs add, a prefix shared by many requests is decoded once with their query rows stacked, and its (A, L) is added to each suffix's in the combine (Appendix D.4).

Depth as a dial. With depth T, cut keys add their weight to L but not their value to A. T is a parameter of each call, not of the cache, so one cache serves every depth, and $T = \infty$ is dense decode. For BF16 values the kernel substitutes a model of each 64-key block, its mean value plus a rank-16 map from key to value deviation fitted on the prompt, and one virtual row per block carries the cut keys' summed weight; for 8-bit values the model is the request's running mean, which needs no fit. We evaluate two finite settings. At $T = 16$, the error is within 1.5% of the lowest BF16 error on six of the seven generations in Section 6. At $T = 14$, the kernel reads fewer bytes while remaining within the BF16 error range on those generations.

At $D = 128$ a BF16 key and value cost 512 bytes. The dense path reads $130 + 128\, r_K + 256$ bytes per key, where r_K is the refined fraction; 8-bit values replace 256 with $128 + 128\, r_V$. At finite depth, BF16 value traffic is $256\, \lambda$ for live fraction λ. On the ablation cells of Section 6.2 these measure 389–423, 261–280, and 196–334 bytes per key. Every configuration stores both planes, 514 bytes per key; the gates save reads.

5 Deterministic backward kernel

Training pairs FlashAttention-4's forward with a FoldAttention backward that keeps the structure of FlashAttention-3's Hopper kernel (a TMA producer warp, two ping-ponged MMA warpgroups, one key block per CTA) and changes how CTAs' partial gradients are added (Figure 4 in Appendix B).

dQ. A preprocess computes $D_i = dO_i^\top o_i$ and the maxima that bound dQ (Equation 4) per request and KV head. Each warpgroup then recomputes $P\, 2^\sigma = \exp2(c\, S - \ell + \sigma)$, placing the grid exponent σ inside the exponential that softmax already requires. The product $dS\, K$ therefore leaves the tensor core on the grid. One `cvt.rni.s32` rounds it, and a store warp adds the tile into a 32-bit global accumulator with a bulk asynchronous reduction. The grid leaves 30 bits for the bound, so the sum cannot overflow (Proposition 2 with $b = 30$). A postprocess scales the integers back (Algorithm 2).

Table 1: Decode on eight-step generations: latency in μs / FP32-relative ℓ_2 error in 10^{-3} (speedup over the fastest BF16 baseline). †: error above the lower end of the BF16 baselines' range.

Model, layer, batch × context	Fastest BF16	BF16 err.	Fold dense	Fold T=16	Fold T=14
Qwen3-30B L24, 8×16K	FA-3 99	1.66–3.00	77 / 1.63 (1.30×)	54 / 1.68† (1.84×)	45 / 2.33† (2.22×)
Qwen3-30B L24, 32×8–16K	FlashInfer 264	1.71–2.33	206 / 1.67 (1.28×)	126 / 1.70 (2.10×)	103 / 2.19† (2.54×)
Qwen3-30B L24, 16×16–32K	FlashInfer 262	1.71–3.13	207 / 1.65 (1.27×)	114 / 1.73† (2.30×)	99 / 2.39† (2.65×)
gpt-oss-20b L9, 32×8–16K	cuDNN 269	1.68–2.83	212 / 1.67 (1.27×)	163 / 1.70† (1.65×)	134 / 2.06† (2.00×)
gpt-oss-20b L21, 32×8–16K	cuDNN 267	1.68–2.72	216 / 1.70† (1.24×)	171 / 1.97† (1.57×)	142 / 2.46† (1.89×)
GLM-4-9B L8, 32×8–16K	FlashInfer 139	1.70–3.05	119 / 1.66 (1.17×)	91 / 1.67 (1.53×)	83 / 1.82† (1.69×)
GLM-4-9B L28, 32×8–16K	FlashInfer 140	1.69–2.95	123 / 1.66 (1.14×)	103 / 1.69† (1.36×)	91 / 2.09† (1.54×)

dK and dV. Keys are shared by the G query heads of a GQA group, so dK and dV also receive partials from several heads. A planner chooses ownership from the request shape. When there is enough parallel work, one CTA walks a key block across all G heads and keeps dK, dV in registers. Wider groups split into subgroups; the last CTA to arrive adds their FP32 partials in subgroup order. When each head has its own CTA, the CTAs instead round their partials onto integer grids, as for dQ.

Scheduling and guarantees. Dense batches run on a persistent kernel that claims tiles from a work list sorted by cost within sections of (request, head) pairs sized to a fixed working-set budget, so heavy causal tiles start first while neighbouring CTAs share their query-side reads in L2. Because no reduction waits on an order, CTAs claim whatever tile is next, and reversing the list leaves every bit unchanged. A request's gradients are the same bits alone or packed with others, because the variable-length plan reads only head counts.

6 Evaluation

Setup. One H100 80GB HBM3, CUDA 13.0, PyTorch 2.13 (Paszke et al., 2019). Baselines are FlashAttention-3 (Shah et al., 2024) and -4 (Zadouri et al., 2026), FlashInfer (Ye et al., 2025) with its tensor-core and TensorRT-LLM XQA decode (NVIDIA Corporation, 2026c), cuDNN (NVIDIA Corporation, 2026a), and DASH (Qiang et al., 2026); for shared prefixes and drafts we add vLLM's cascade path, PAT (Yi et al., 2026), FastTree (Pan et al., 2025), and SGLang's tree verification (Zheng et al., 2024) (versions in Appendix C). Each library runs every configuration it offers at each shape and is represented by its fastest; every kernel is scored against one FP32 reference. We replay CUDA graphs with L2 evicted before each sample and the kernel order rotated each round, and report medians of paired per-round ratios.

6.1 Decode on model generations

We capture post-RoPE (Su et al., 2024) queries, keys, and values from prefills of Qwen3-30B-A3B (layer 24, $D = 128$, $G = 8$), gpt-oss-20b (layers 9 and 21, $D = 64$, $G = 8$), and GLM-4-9B (layers 8 and 28, $D = 128$, $G = 16$), decode eight steps with the model's own next tokens, and time every kernel on the final state (Table 1; Figure 5 plots latency against error along the depth dial). Dense FoldAttention is faster than every baseline in all seven cases: 1.24–1.30× at $G = 8$ and 1.14–1.17× at $G = 16$. Its error is within 1.2% of the lowest BF16 error and below it in six cases. At depth 16, decode is 1.36–2.30× faster, with an error within 1.5% of the lowest BF16 error in six cases and 17% above it on gpt-oss-20b layer 21. Depth 14 is 1.54–2.65× faster at 1.07–1.46× that error, still below XQA's in every case. The front kernel adds 4–8 μs per step.

6.2 Ablation

Table 2 adds one mechanism at a time. Reading both key planes costs the same bytes as a BF16 cache, and in this configuration FoldAttention, despite its extra refinement product, runs at 0.98–1.01× the fastest vendor-tuned BF16 kernel. Its speedup comes from the reads that final weights eliminate. Gating the second plane refines 2–29% of keys at $D = 128$, reduces traffic by 18–24%, and yields 1.18–1.27× speedup.

Table 2: Decode kernel speed over the fastest BF16 baseline's decode and bytes read per key, adding one mechanism at a time (ragged batches of 256 KV heads, 16K context). Errors are FP32-relative ℓ_2 in 10^{-3}; the most accurate BF16 baseline's is 1.67–1.69 in these cells, and XQA's 2.31–2.83.

	Qwen D=128, G=8	Qwen D=128, G=4	GLM D=128, G=16	gpt-oss D=64, G=8	Error
Both planes	0.98× (514 B)	0.99× (514 B)	0.98× (514 B)	1.00× (258 B)	1.64–1.66
Gated plane B (dense)	1.26× (390 B)	1.27× (389 B)	1.18× (423 B)	1.24× (204 B)	1.66–1.67
+ 8-bit values	1.76× (262 B)	1.79× (261 B)	1.41× (275 B)	1.56× (134 B)	1.68–1.70
+ depth T=16, BF16 values	2.15× (217 B)	2.37× (196 B)	1.53× (308 B)	1.65× (141 B)	1.70–1.72

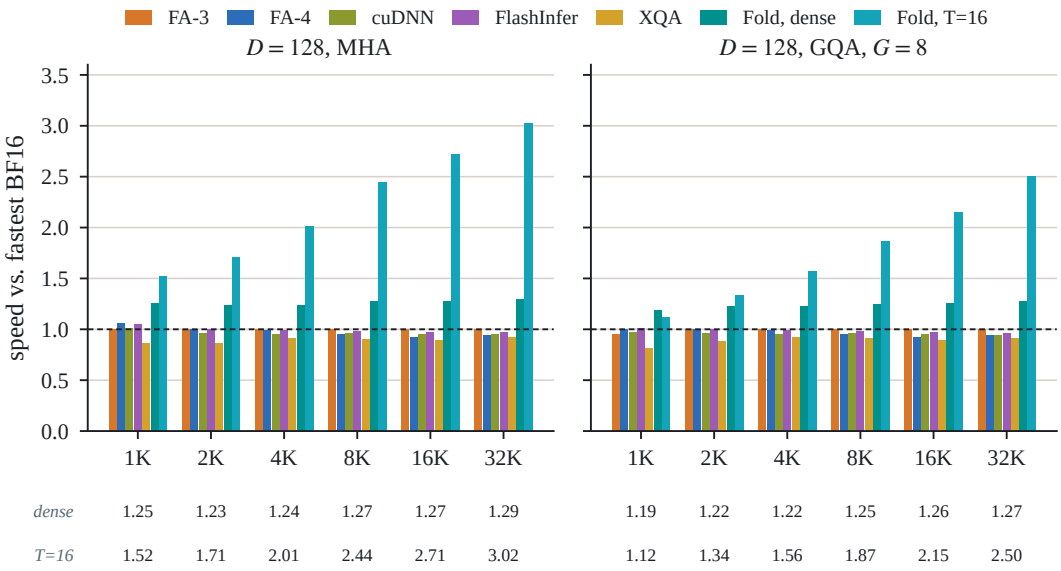

Figure 3: Decode speed relative to the fastest BF16 baseline in each cell (dashed line at 1), on ragged batches of 256 KV heads at $D = 128$, for MHA and GQA with $G = 8$. The table under each panel gives FoldAttention's speedups, dense and at depth 16.

Why the reference is declared. Online softmax could gate the same reads against its running maximum, which bounds each final weight from above. Emulated on the seven generations with the kernel's logits, gates, and splits (Appendix D.6), that policy reads 481–510 bytes per key at $D = 128$, against 390–426 under the declared reference, because each split restarts its maximum. Even with one split per request, it reads 3–16% more in dense decode and 1.2–1.7× as many bytes at depth 14. At the BF16 kernels' error, no other method we emulate (Quest (Tang et al., 2024), Faster Flash Decoding, KIVI, INT8 and FP8 caches) reads fewer bytes than a BF16 kernel; depth 14 reads 117–279.

6.3 Context and group-size sweep

Figure 3 sweeps context from 1K to 32K keys for MHA and GQA groups of eight at $D = 128$, using ragged batches of 256 KV heads. Dense FoldAttention is faster than every baseline in every cell: 1.19–1.25× at 1K and 1.27–1.29× at 32K, where it reads at 95–97% of the measured bandwidth ceiling. MHA gains most, because no verdict is ORed across a group. At 32K, depth 16 is 3.02× faster at $D = 128$ and 2.83× faster at $D = 64$. Across all 84 cells of both head dimensions, groups of one to sixteen, and ragged and uniform batches (Appendix D), dense decode reaches 1.35× and is slower only once (0.95× for uniform $D = 64$, $G = 8$ at 1K). Depth 16 reaches 3.09× while remaining within the BF16 error range in 82 cells. Depth 14 reaches 3.50× and remains within that range in 67. Dense decode with 8-bit values reaches 1.89×.

Table 3: Qwen3-8B decoding for real in each arm. KL is FlashAttention-3's next-token distribution against the arm's, in 10^{-4} nats per token, and ΔNLL the arm's teacher-forced negative log-likelihood minus FlashAttention-3's, in 10^{-4} nats, on 32 WikiText-103 documents of 1024 tokens. RULER is the mean exact match over four tasks. Contexts past 32K use YaRN (Peng et al., 2024); their cuDNN and 8-bit V cells come from a second run, in which FlashAttention-3 scores 70.3 on RULER. "Diverge" is the median position at which greedy generation first leaves FlashAttention-3's.

Arm	KL, 8K / 16K / 32K			ΔNLL 32K	RULER 8–32K	KL 64K / 128K	RULER 64–128K	LongBench 16 tasks	Diverge
FA-3	0	0	0	0	87.1	0 / 0	68.0	49.3	–
FlashInfer	7.9	8.0	8.6	−2.9	86.9	9.1 / 9.8	66.8	49.3	55
cuDNN	8.0	8.1	8.9	−1.1	87.1	8.8 / 10.2	69.9	–	61
FlashInfer FP8	73.9	109.0	131.8	+75.1	87.3	124.8 / 174.6	66.0	49.2	26
Fold dense	8.1	8.1	8.5	+0.2	87.6	8.9 / 10.5	67.6	49.3	90
Fold dense, 8-bit V	8.0	8.0	8.1	+0.1	87.1	9.0 / 9.6	69.5	–	78
Fold T=16	8.1	8.1	8.2	+0.0	86.6	8.9 / 10.3	68.0	49.3	49
Fold T=14	8.1	8.1	8.5	+4.3	86.9	9.1 / 9.8	67.6	49.2	67

6.4 Serving and quality

We decode Qwen3-8B (Yang et al., 2025) ($G = 4$, $D = 128$) with the whole model step captured in one CUDA graph, changing only the attention call and its cache, at 32 requests of 8K, 16 of 16K, and 8 of 32K. Dense FoldAttention makes the whole step 1.13–1.14× faster. Depth 16 gives 1.35–1.46×, and depth 14 gives 1.44–1.54×. With BF16 values, a finite depth needs the block model, fitted on the host in 6.6–11.3 ms per layer per prefill, which the depth-16 step recovers within 38–65 generated tokens; with 8-bit values it costs nothing (Appendix D).

Each arm decodes into its own cache and is teacher-forced after 8K–32K prompts (Table 3). FoldAttention's KL divergence from BF16 FlashAttention-3 is 8.0–8.5$\times 10^{-4}$ nats per token, compared with 7.9–8.9$\times 10^{-4}$ for FlashInfer and cuDNN, and its likelihood differs by at most 5.5×10^{-4} nats. The FP8 cache has 9–16× that divergence and a likelihood up to 7.5×10^{-3} nats worse. At 64K and 128K, FoldAttention remains at FlashInfer's divergence level (8.9–10.5 versus 9.1–9.8$\times 10^{-4}$), while FP8 reaches 175×10^{-4}. Its mean RULER accuracy over four tasks is 67.6–68.0%, against 68.0% for FlashAttention-3, 66.8% for FlashInfer, and 66.0% for FP8. On the 16 English tasks of LongBench (Bai et al., 2024), its mean score is 49.2–49.3 against 49.3 for FlashAttention-3. Greedy generations of 1024 tokens first leave FlashAttention-3's after a median of 49–90 tokens with FoldAttention, 55–61 with the other BF16 kernels, and 26 with FP8.

6.5 Cascades and drafts

On 16 shared-prefix cascades of Qwen3-30B-A3B and gpt-oss-20b, dense FoldAttention is faster than every kernel we ran in every cell, 1.04–1.42× over the fastest, and 1.34×, 1.32×, and 1.61× faster by geometric mean than vLLM's cascade path, PAT, and FastTree; on prefix trees it matches PAT (1.07×). Verifying chains of two or four drafts as extra query columns, it is 0.99–1.24× as fast as the fastest library; longer chains and trees favor FA-3 and SGLang (Appendix D.7).

6.6 Backward

On eight GQA and MHA model shapes at 16K tokens or more (Table 4), FoldAttention is 1.29× and 1.23× faster than deterministic FlashAttention-3 and -4, 1.32× faster than DASH, and 1.05× faster than the fastest nondeterministic kernel; packed variable-length batches behave the same. On the grid FlashAttention-3 and DASH report, it is 1.12× faster than DASH with MHA, and with GQA groups of eight up to 1.84× faster than deterministic FlashAttention and 1.87× faster than DASH (Appendix E). In the same kernel, FP32 atomics for dQ are 1.0% faster over 32 shapes, while replacing the persistent work list with one tile per CTA is 3.9% slower. The order-free sum is what lets a deterministic kernel claim tiles in any order; the deterministic modes of FlashAttention-3 and -4 take 24% and 19% longer than their nondeterministic modes.

Determinism and training. FoldAttention's gradients are bit-identical when a request is rerun, batched, or packed, as are those of the deterministic modes of FlashAttention-3 and -4 and of DASH,

Table 4: Backward speedup of FoldAttention over each baseline (geometric mean). "Fastest" takes the best kernel of its kind at each shape. Training time combines FA-4's forward with each backward.

Set	FA-3	FA-3 det.	FA-4	FA-4 det.	cuDNN	DASH	Fastest det.	Fastest nondet.
Model shapes, backward (8)	1.07	1.29	1.06	1.23	1.23	1.32	1.23	1.05
Packed varlen, backward (5)	1.08	1.30	1.07	1.29	–	–	1.28	1.06
MHA grid, backward (24)	0.99	1.18	0.98	1.11	1.17	1.12	1.07	0.97
GQA grid (G=8), backward (24)	1.06	1.32	1.04	1.25	1.23	1.28	1.23	1.04
Model shapes, train (8)	0.99	1.15	1.01	1.14	1.26	1.17	1.12	0.99

while the nondeterministic kernels repeat their bits on at most 8% of shapes (Appendix F). On operands captured from training, FoldAttention's dQ, dK, and dV match FlashAttention-3's ℓ_2 error to three digits; their per-element relative errors are also comparable through the 99th percentile (Table 7). With FlashAttention-4's forward, a forward and backward step is 1.12× faster than the fastest deterministic alternative, and a whole-model training step of a 1B Llama (Grattafiori et al., 2024) is within 0.5% of the nondeterministic kernels. Three FoldAttention runs trained from scratch for 2000 steps of 16K tokens produce bit-identical weights. Their validation loss is 3.946, within the 3.939–3.951 range of the other kernels (Figure 13).

7 Related work

Attention kernels and state merges. Hydragen, vLLM's cascade path, PAT, and FastTree read a shared prefix once and merge its online-softmax state into each request's (Juravsky et al., 2024; Kwon et al., 2023; Yi et al., 2026; Pan et al., 2025), as SGLang does for draft trees (Zheng et al., 2024); FoldAttention adds the levels' pairs instead.

Fixed softmax references. FlashDecoding++ applies one profiled constant, with a synchronized fallback when scores leave its FP16 range (Hong et al., 2024); FlashAttention-4 skips small rescales (Zadouri et al., 2026); VFA freezes a maximum from block summaries (Sun et al., 2026); and Faster Flash Decoding and BLASST skip blocks below a pseudo- or running maximum, discarding their mass (Liu et al., 2026; Yuan et al., 2025). FoldAttention needs only a reference in the exponent range, and uses final weights for per-key reads, additive merges, and a dense denominator at every depth.

Quantized caches and sparse decode. KIVI, KVQuant, and FP8 caches compress the KV cache (Liu et al., 2024; Hooper et al., 2024; Micikevicius et al., 2022), and Quest, SparQ, and CoSA select pages or skip value reads by approximate scores (Tang et al., 2024; Ribar et al., 2023; Xue et al., 2026). These methods primarily evaluate task accuracy rather than agreement with a BF16 kernel. Among the methods we emulate, none reads fewer bytes than a BF16 kernel at its output error (Section 6.2).

Reproducible reduction. Pre-rounding summands to a common grid makes floating-point sums reproducible (Demmel & Nguyen, 2013; Ahrens et al., 2020). DASH reschedules FlashAttention's ordered backward as a DAG (Qiang et al., 2026), and batch-invariant kernels fix reduction topologies and split sizes for serving and for reinforcement learning, whose rollouts must match the trainer (Zhang et al., 2025; He & Thinking Machines Lab, 2025; The SGLang Team, 2025; Zhong et al., 2026). FoldAttention pre-rounds inside the attention backward with bounds it computes cheaply, so determinism needs no ordering.

8 Limitations and conclusion

The kernels target Hopper, head dimensions 64 and 128 for decode (96 also for backward), and MHA and GQA caches; latent caches such as MLA's are not yet supported, and porting the WGMMA layouts to Blackwell is future work. Online softmax discovers a row's reference; FoldAttention declares it. Final weights let decode read only the bytes they need, from BF16-kernel accuracy to three times faster on one cache, and a declared grid lets the backward sum integers in any order, deterministic and faster than the nondeterministic kernels.

Reproducibility statement

Section 3 states the numerical contract, and Sections 4 and 5 describe the kernels. Appendix C specifies the baseline configurations, the timing protocol, and the accuracy reference. The supplementary material holds the kernels, written in CuTe DSL (NVIDIA Corporation, 2026b), the benchmark suite, and every result file with its raw timing samples; all will be released as open source. The attention inputs are regenerated from the public model weights.

The use of large language models

Generative AI assisted literature retrieval, kernel and benchmark ideation, source-code review, figure generation, and drafting and editing this manuscript. The authors are responsible for the algorithms, code, experiments, citations, and final claims, including verification against the implementation and recorded results.

References

Peter Ahrens, James Demmel, and Hong Diep Nguyen. Algorithms for efficient reproducible floating point summation. *ACM Transactions on Mathematical Software*, 46(3), 2020.

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

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *Advances in Neural Information Processing Systems*, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

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*, 2022.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference. `https://crfm.stanford.edu/2023/10/12/flashdecoding.html`, 2023.

James Demmel and Hong Diep Nguyen. Fast reproducible floating-point summation. In *IEEE Symposium on Computer Arithmetic (ARITH)*, 2013.

Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Horace He and Thinking Machines Lab. Defeating nondeterminism in LLM inference. Thinking Machines Lab: Connectionism, 2025. `https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/`.

Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2 edition, 2002.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. FlashDecoding++: Faster large language model inference on GPUs. In *Proceedings of Machine Learning and Systems*, 2024.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*, 2024.

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.

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*, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, 2023.

Zhigeng Liu, Zhiyuan Ning, Ruixiao Li, Xiaoran Liu, Yuerong Song, Min Zhang, Ziwei He, and Xipeng Qiu. Faster than flash: Exploiting attention sparsity for efficient long-context decoding. *arXiv preprint arXiv:2609.00097*, 2026.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning*, 2024.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

NVIDIA Corporation. NVIDIA cuDNN: Scaled dot product attention. `https://docs.nvidia.com/deeplearning/cudnn/latest/operations/Attention.html`, 2026a. Accessed September 19, 2026.

NVIDIA Corporation. CUTLASS and CuTe DSL. `https://github.com/NVIDIA/cutlass`, 2026b.

NVIDIA Corporation. TensorRT-LLM. `https://github.com/NVIDIA/TensorRT-LLM`, 2026c.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Zaifeng Pan, Yitong Ding, Yue Guan, Zheng Wang, Zhongkai Yu, Xulong Tang, Yida Wang, and Yufei Ding. FastTree: Optimizing attention kernel and runtime for tree-structured LLM inference. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, 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*, 2019.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *International Conference on Learning Representations*, 2024.

Xinwei Qiang, Hongmin Chen, Shixuan Sun, Jingwen Leng, Xin Liu, and Minyi Guo. DASH: Deterministic attention scheduling for high-throughput reproducible LLM training. *arXiv preprint arXiv:2601.21824*, 2026.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ Attention: Bandwidth-efficient LLM inference. *arXiv preprint arXiv:2312.04985*, 2023.

Rya Sanovar, Srikant Bharadwaj, Renee St. Amant, Victor Rühle, and Saravan Rajmohan. Lean attention: Hardware-aware scalable attention mechanism for the decode-phase of transformers. *arXiv preprint arXiv:2405.10480*, 2024.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08608*, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568, 2024.

Yupeng Sun, Yanzhao Li, Zhiqiang Zou, Bai Du, Zhiyuan Zhang, Hui Dong, Gaoyige Fan, and Hui Wang. Vfa: Relieving vector operations in flash attention with global maximum pre-computation. *arXiv preprint arXiv:2604.12798*, 2026.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Team GLM. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

The SGLang Team. Towards deterministic inference in SGLang and reproducible RL training. LMSYS Org Blog, 2025. `https://lmsys.org/blog/2025-09-22-sglang-deterministic/`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

Yufei Xue, Lin Niu, Hong Liu, Siran Liu, Hanyong Shao, Wei Liu, Guanghua Yu, Jianchen Zhu, and Jun Zhang. CoSA: Accelerating long-context inference via proxy-kernel co-designed sparse attention. *arXiv preprint arXiv:2607.25291*, 2026.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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. *arXiv preprint arXiv:2501.01005*, 2025.

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 (ASPLOS)*, 2026.

Jiayi Yuan, Cameron Shinn, Kai Xu, Jingze Cui, George Klimiashvili, Guangxuan Xiao, Perkz Zheng, Bo Li, Yuxin Zhou, Zhouhai Ye, Weijie You, Tian Zheng, Dominic Brown, Pengbo Wang, Markus Hoehnerbach, Richard Cai, Julien Demouth, John D. Owens, Xia Hu, Song Han, Timmy Liu, and Huizi Mao. BLASST: Dynamic BLocked attention sparsity via softmax thresholding. *arXiv preprint arXiv:2512.12087*, 2025.

Ted Zadouri, Markus Hoehnerbach, Jay Shah, Timmy Liu, Vijay Thakkar, and Tri Dao. FlashAttention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling. *arXiv preprint arXiv:2603.05451*, 2026.

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. *arXiv preprint arXiv:2511.17826*, 2025.

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 (NeurIPS)*, 2024.

Tianle Zhong, Neiwen Ling, Yifan Pi, Zijun Wei, Tianshu Yu, Geoffrey Fox, Peng Wu, and Xiao Yu. Diagnosing training inference mismatch in LLM reinforcement learning. *arXiv preprint arXiv:2605.14220*, 2026.

A Proofs

Proof of Proposition 1. (i) Both components of F_i are sums over keys of terms that depend only on s_{ij}, v_j, and Z_i, so a sum over $I \cup J$ splits into the sums over I and J when they are disjoint. (ii) Multiplying numerator and denominator by $2^{Z_i} > 0$,

$$\frac{A_i}{L_i} = \frac{\sum_j 2^{s_{ij}-Z_i} v_j}{\sum_j 2^{s_{ij}-Z_i}} = \frac{\sum_j 2^{s_{ij}} v_j}{\sum_j 2^{s_{ij}}} = \sum_j \mathrm{softmax}_j\big(q_i^\top k_j/\sqrt{D}\big)\, v_j,$$

since $2^{s_{ij}} = e^{q_i^\top k_j/\sqrt{D}}$. The quotient does not depend on Z_i. (iii) Every weight satisfies $0 \le 2^{s_{ij}-Z_i} \le 2^{m_i-Z_i}$, so the exact sums obey $L_i \le n\,2^{m_i-Z_i} < 2^{127}/\nu$ and $|A_{id}| \le \nu L_i < 2^{127}$. Recursive FP32 summation of nonnegative terms with unit roundoff $u = 2^{-24}$ returns at most $(1+u)^n$ times the exact sum (Higham, 2002), and $(1+u)^{2^{22}} < e^{1/4} < 2$; the same factor bounds $\sum_j 2^{s_{ij}-Z_i}|v_{jd}|$, which dominates the computed $|A_{id}|$. Every intermediate therefore stays below 2^{128}, the FP32 and BF16 overflow threshold. With flush to zero, a weight is lost only if it is below 2^{-126}, so at most n weights totalling less than $n\,2^{-126}$ are lost, while $L_i \ge 2^{m_i-Z_i}$ from the largest weight alone. The relative change of L_i is at most $n\,2^{-126-(m_i-Z_i)}$. □

On the generations of Section 6, $m_i - Z_i$ lies in $[-8.2, 16.7]$ (Figure 9). With $n \le 2^{22}$ and values below 2^{16}, the left side of the overflow condition is at most 55, far below 127, and the flush bound is below 2^{-95}.

Proof of Proposition 2. Round to nearest moves each term by at most $\frac{1}{2}$, so $|\hat{g}_c| \le \alpha|g_c| + \frac{1}{2}$. Because $\lceil \log_2 B \rceil \ge \log_2 B$, $\alpha \le 2^b/B$, and

$$\textstyle\sum_c |\hat{g}_c| \le \alpha \sum_c |g_c| + \frac{C}{2} \le 2^b + \frac{C}{2} < 2^{b+1}.$$

Any partial sum, over any subset of the $\hat{g}_c$ in any order, is bounded by the same quantity and so is representable in $(b+2)$-bit two's complement, whose range is $[-2^{b+1}, 2^{b+1})$. Integer addition without overflow is exact, hence associative and commutative, so every order and grouping gives $\sum_c \hat{g}_c$. For the error, $|\alpha^{-1}\sum_c \hat{g}_c - \sum_c g_c| \le \alpha^{-1}\sum_c |\hat{g}_c - \alpha g_c| \le C/(2\alpha)$, and $\lceil \log_2 B \rceil < \log_2 B + 1$ gives $\alpha > 2^{b-1}/B$, so $C/(2\alpha) < C\,B\,2^{-b}$. □

In the kernel, g_c is the FP32 partial one CTA computes, with the power of two α already applied inside the exponential, which is exact barring underflow. The bound covers every partial because Equation 4 bounds the sum of the absolute values of all terms of the element, of which each partial is a subset, and the 2^{-6} widening covers the BF16 rounding of P and dS that the computed partial carries. The dQ grid uses $b = 30$ with 32-bit accumulators; the dK grid, and the dV grid when $G \cdot S > 16384$, use $b = 61$ with 64-bit accumulators.

B Algorithms

Algorithm 1 gives one decode CTA's loop over the 64-key tiles of its split for one GQA row group, with BF16 values and two weight terms; 8-bit values add a gated second value plane, a finite depth with the block tail adds a virtual row per block, and the 128-key tile holds two keys in each row of the logit (Section 4). Query row i has INT8 planes q_i^a, q_i^b and scale η_i, key j has planes a_j, b_j and scale e_j, and scores are in base 2.

Algorithm 1 FoldAttention decode, one CTA

Require: references Z_i, depth T (∞ for dense), refine gate τ_K, key tiles $1, \ldots, N$ of this split, every n-th tile of the request for n splits

$A \leftarrow 0 \in \mathbb{R}^{D\times G}$, $L \leftarrow 0 \in \mathbb{R}^{G}$ ▷ FP32
issue a bulk copy of tile 1's plane A and key scales
for $t = 1, \ldots, N$ **do**
  wait for tile t's plane A; issue tile $t{+}1$'s
  $c^{aa}, c^{ab} \leftarrow K_t^a\,[Q^a; Q^b]^\top$ ▷ INT8 WGMMA, keys on M
  $\tilde{s}_{ij} \leftarrow \eta_i e_j\,(c^{aa}_{ji} + c^{ab}_{ji}/256) - Z_i$ ▷ coarse score against the fixed reference
  $\text{live}_j \leftarrow \bigvee_i[\tilde{s}_{ij} \geq -T]$; $\text{refine}_j \leftarrow \text{live}_j \wedge \bigvee_i[\tilde{s}_{ij} \geq -\tau_K]$ ▷ ORed over the group
  if no key is live and there is no block tail **then**
    continue
  end if
  gather b_j for refined keys and v_j for live keys
  $c^{ba}, c^{bb} \leftarrow K_t^b\,[Q^a; Q^b]^\top$ ▷ second INT8 WGMMA
  $s_{ij} \leftarrow \tilde{s}_{ij} + [\text{refine}_j]\,\eta_i e_j\,(c^{ba}_{ji} + c^{bb}_{ji}/256)/256$
  $p_{ij} \leftarrow [\text{live}_j \wedge s_{ij} \geq -T]\;2^{s_{ij}}$ ▷ final weight, FP32
  add the weights of cut keys to the tile's virtual row ▷ finite depth only
  $P^{hi} \leftarrow \text{bf16}(p)$; $P^{lo} \leftarrow \text{bf16}(p - P^{hi})$
  $A^\top \leftarrow A^\top + V_t^\top P^{hi\top} + V_t^\top P^{lo\top}$ ▷ BF16 WGMMA, channels on M
  $L_i \leftarrow L_i + \sum_j p_{ij}$
end for
write (A, L) to this split's slot ▷ combine: $o_i = \sum_{\text{slots}} A_i \big/ \sum_{\text{slots}} L_i$ in slot order

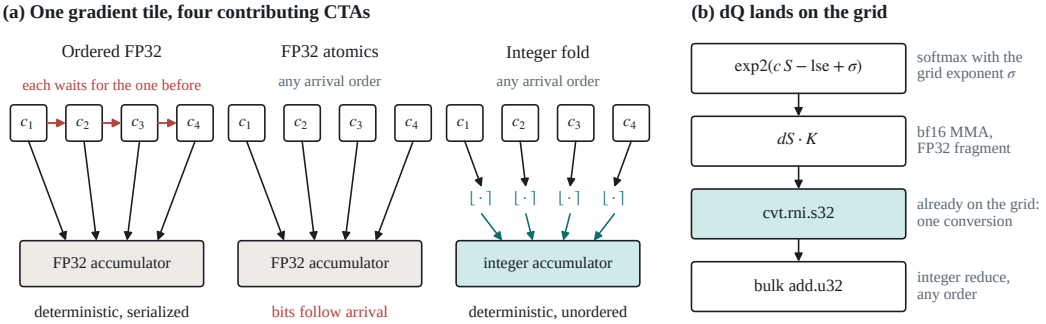

Figure 4: *(a)* Four CTAs' partials into one gradient tile: ordered FP32 adds wait for each other, atomics follow arrival order, and the integer fold rounds each partial onto one grid ($\lfloor\cdot\rceil$) and adds in any order. *(b)* The grid exponent σ rides the softmax exponential, so dQ partials leave the tensor core on the grid.

Algorithm 2 gives the backward for one work-list tile, a key block n of one head. A preprocess has computed $D_i = dO_i^\top o_i$, the base-2 log-sum-exp ℓ_i, and the grid exponent $\sigma = \log_2 \alpha$ of the tile's request and KV head from Equation 4; c is the softmax scale.

Algorithm 2 FoldAttention backward, one key block

Require: $Q, dO \in \mathbb{R}^{S\times D}$, K_n, V_n, ℓ, D, grid exponent σ

load K_n, V_n; $dK_n, dV_n \leftarrow 0$ ▷ FP32 registers
for each query block m in the causal range of n **do**
  $S \leftarrow Q_m K_n^\top$; $\tilde{P} \leftarrow \exp_2\big(c\log_2(e)\, S - \ell_m + \sigma\big)$ ▷ $\tilde{P} = 2^\sigma P$
  $dV_n \leftarrow dV_n + \tilde{P}^\top dO_m$
  $dP \leftarrow dO_m V_n^\top$; $d\tilde{S} \leftarrow \tilde{P} \circ (dP - D_m)$
  $\widehat{dQ} \leftarrow \mathrm{RNE}_{s32}(d\tilde{S}\, K_n)$ ▷ already on the grid
  bulk integer add of $\widehat{dQ}$ into dQ_m^{acc} ▷ order-free
  $dK_n \leftarrow dK_n + d\tilde{S}^\top Q_m$
end for
if this CTA owns the key block's whole group **then**
  store $c\,2^{-\sigma} dK_n$ and $2^{-\sigma} dV_n$
else if the group is split into subgroups **then**
  write the FP32 partial; the last subgroup to arrive sums all partials in subgroup order and stores
else
  round dK_n, dV_n onto their integer grids and bulk-add them
end if
postprocess: $dQ \leftarrow c\,2^{-\sigma} dQ^{\text{acc}}$, and likewise for integer dK, dV

C Experimental details

Baseline configurations. We use FlashAttention-3 3.0.0, FlashAttention-4 4.0.0b31, FlashInfer 0.7.0, cuDNN 9.26, DASH at commit d87bcc9, vLLM 0.30.0, PAT at commit 8cb067f, and sglang-kernel 0.4.7. Each library runs every configuration it offers at a shape: paged and contiguous caches, packed GQA, split counts 1–16 for FlashAttention-3, the tensor-core and CUDA-core FlashInfer backends, and cuDNN through FlashInfer's paged binding and through PyTorch SDPA. Configurations that fail are dropped, the rest are timed over nine rounds, and each library is represented by its fastest.

Timing. The selected baselines and FoldAttention are then timed together by CUDA-graph replay, with the L2 cache evicted by reading a 256 MB buffer before every sample and the kernel order rotated with a stride coprime to the number of kernels. We run at least 21 rounds, drop the first, and report medians of per-round paired ratios.

Accuracy reference. Decode outputs are scored against FP32 attention over the same BF16 inputs, and backward gradients against FP32 gradients computed one head at a time without TF32. A kernel whose error exceeds 20 times the best error at its precision is treated as computing a different function and excluded from every ratio.

DASH. DASH and FlashAttention-3 register the same PyTorch operators and cannot be loaded together. DASH therefore runs in its own process with FlashAttention-4 and FoldAttention, and its times are joined to the FlashAttention-3 process through FoldAttention's time in each. FlashAttention-4's ratio to FoldAttention agrees between the two processes to within 1% at 59 of 68 shapes and 2.5% at every shape, which bounds the join's error. DASH was built from commit d87bcc9 with a one-line patch restoring FlashAttention-3's masking bound for the diagonal block of its reversed causal loop; without it, DASH's head-dim-64 causal gradients have 15–60$\times$ the error of every other kernel. DASH has no variable-length schedule and is measured on dense batches only.

Shared-prefix baselines. vLLM's cascade path runs FA-3 over the prefix the whole batch holds, FA-3 over each request's remainder, and a state merge, so a prefix tree cascades over its root. PAT launches on the legacy default stream, which a CUDA graph cannot capture, and is timed eagerly with the same L2 eviction. At $D = 64$ with $G = 8$, its output fails the accuracy criterion even on random inputs, and one run ended in an illegal memory access. We therefore report no gpt-oss-20b cells. On Qwen3-30B-A3B, the accuracy rule drops it in one cascade cell, and in two others its error is 7–8 times the BF16 kernels'; without those two, its cascade range is 1.18–1.48$\times$. FastTree fails the accuracy criterion on three single-prefix cascades, which are omitted.

D Additional decode results

D.1 Latency against error on every generation

Figure 5 plots latency against error for every generation of Table 1, along the depth dial.

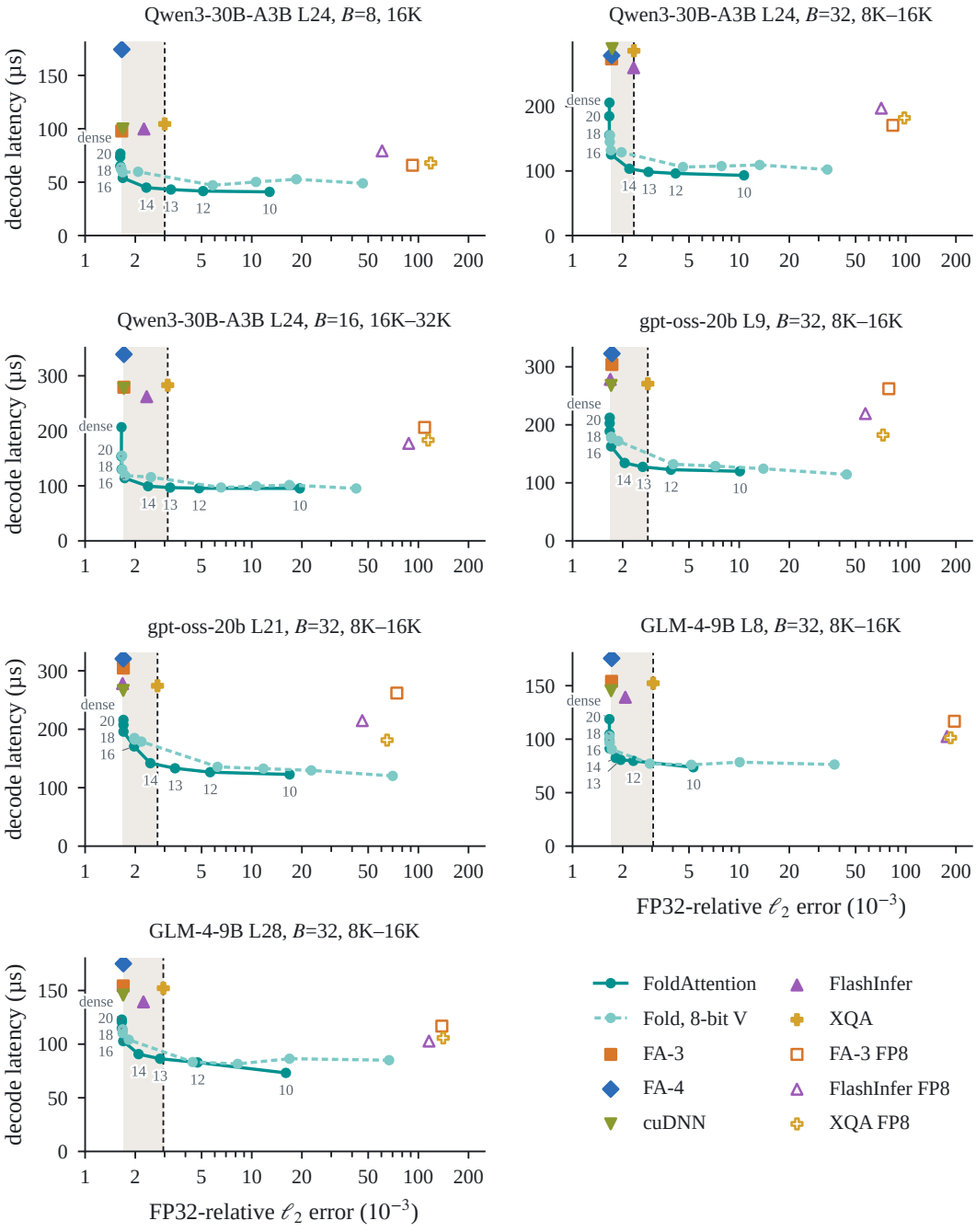

Figure 5: Decode latency against FP32-relative error on the seven generations of Table 1. FoldAttention is a curve over depth T, BF16 (solid) or 8-bit (dashed) values, with each point's T labeled on the BF16 curve; the 8-bit curve's points are the same depths in the same order. Libraries are points, FP8 caches hollow. The shaded band spans the BF16 kernels' errors. On gpt-oss-20b layer 21, no FoldAttention variant meets the most accurate BF16 kernel's error.

D.2 Context sweeps

Figure 6 extends the sweep of Figure 3 to every measured group size and adds depth 14.

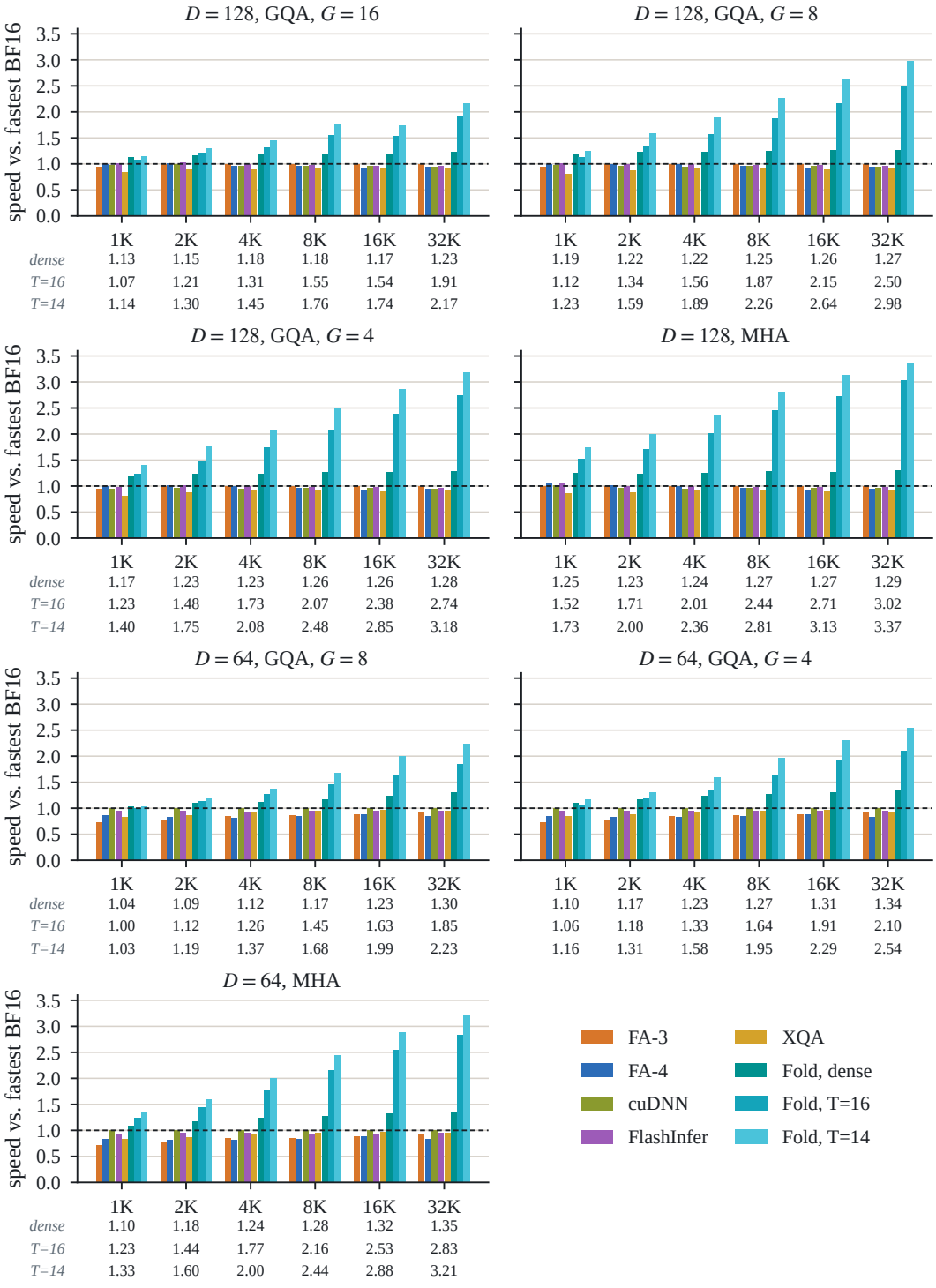

Figure 6: The sweep of Figure 3 for every measured group size (ragged batches of 256 KV heads). At $D = 64$, groups of one and four run on the 128-key tile.

Figure 7 repeats the context sweep on uniform batches, where FlashInfer or cuDNN is usually the fastest baseline and dense FoldAttention is 0.95× at $D = 64$, $G = 8$ and 1K context and 1.03–1.31× elsewhere.

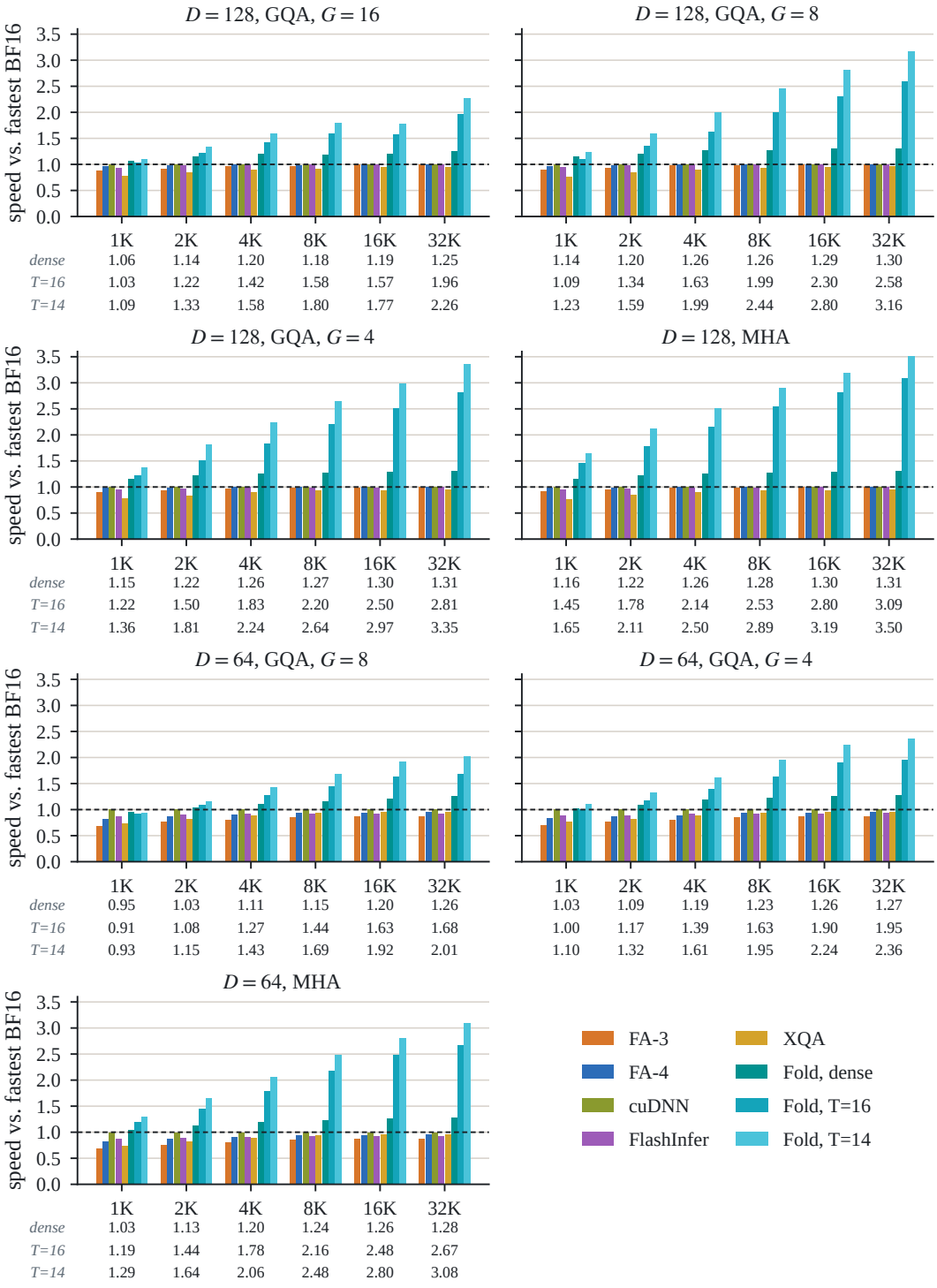

Figure 7: The sweep of Figure 6 on uniform batches.

Figure 8 stores values in two E4M3 planes, which only the 64-key tile reads; dense decode with 8-bit values reaches 1.59–1.86× at 32K.

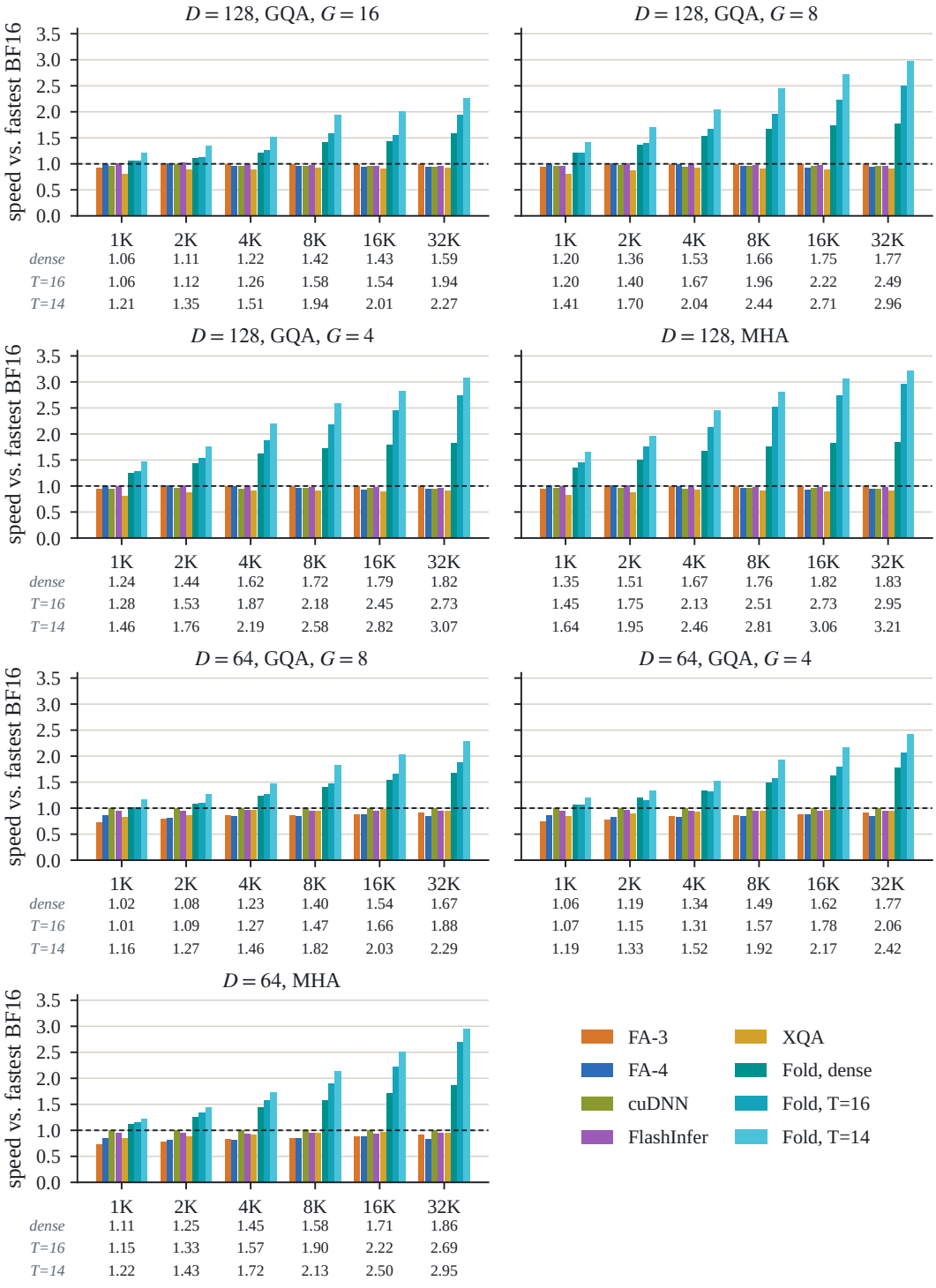

Figure 8: The sweep of Figure 6 with 8-bit values. "Dense" reads every value row in two E4M3 planes; baselines read BF16 caches.

D.3 Reference quality

Figure 9 shows how far the estimate of Section 3 lands from the truth over every row of the seven generations: the true log-sum-exp sits -0.4 to 17.2 binades above Z_i, 1.0 at the median. Decoding Qwen3-8B for real at 64K and 128K context with YaRN (Section 6.4), it sits -0.9 to 28.2 binades above Z_i. Over all 1.8 billion rows decoded in Qwen3-8B at 8K–128K, including LongBench, 118 fall below the certificate's window $[2^{-1}, 2^{100}]$, each with Z_i 1.0–1.6 binades above the log-sum-exp; none exceeds it.

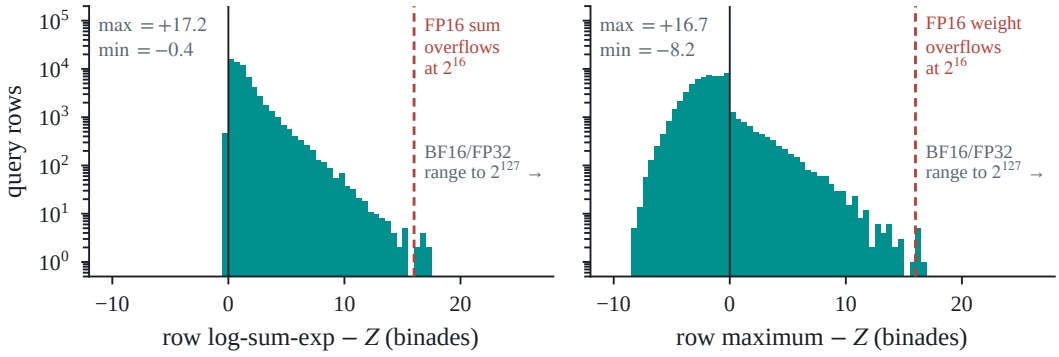

Figure 9: How far each row's true log-sum-exp (left) and largest score (right) sit above the estimated Z_i, over 63,488 rows of the seven generations. FP16 weights would overflow at the right edge; BF16 and FP32 extend to $+127$.

A better estimate would buy little. We drive one kernel-level call with two references: the mass estimate of Section 3, and each row's exact log-sum-exp of its FP32 scores. Refine gates, weight terms, the cut model (the running mean value) and depths 10–20 are the same for both; only Z_i differs. Cells are the four of Table 2 at 4K and 16K on uniform batches of 256 KV heads. Because the estimate sits below the log-sum-exp, a given depth keeps more keys under it and has lower error, so we compare at matched error: for each point of the estimate we interpolate the oracle's bytes per key, linear in log error between adjacent depths. At depths 12–14 the oracle needs 0.94–0.99 of the estimate's bytes (0.97 by geometric mean), with its largest saving on GLM-4-9B at 16K. In dense decode the two read within 3% of each other's bytes and within 1.1% of each other's error.

D.4 Tile layouts

At $D = 64$, a group of up to four rows with BF16 values leaves most of the product's N idle, while each tile pays its copies, round trips, and barriers for 64 keys. These groups walk 128-key tiles: each row of M holds two keys of plane A as stored, and each query takes two columns, $[q \mid 0]$ and $[0 \mid q]$, in the idle part of N. The products are those of two 64-key tiles, so the logits are the same bits, and each copy, round trip, and barrier serves 128 keys. At $D = 128$ the wider tile halves the CTAs an SM holds and gains nothing.

A cascade level stacks the query rows of the requests that hold it, and up to 64 rows fit the decode kernel's N. Past 64 they fill M on their own, and the level runs on a kernel laid out as prefill is, with 64 query rows and 64 keys per tile and two consumer warpgroups taking turns at the tensor cores. Its logit is one FP16 product of $\mathrm{fp16}(a_j + b_j/256)$, rounded once per prefix, against the rows' queries; 11 bits put each logit within about 2^{-11} of $|q||k|$, below the BF16 rounding of the weight. Draft batches past 64 rows run on the same kernel.

D.5 Cache construction

Writing a prompt into the dense cache takes 0.18–0.25 ms per layer (0.6–11% of FlashAttention-4's prompt attention) against 0.05–0.09 ms for a BF16 page append. A finite depth with BF16 values also needs the block model, which the host fits in 6.6–11.3 ms per layer per prefill, 0.2 to 5.3 times the prompt's attention from 32K down to 2K tokens. With 8-bit values the cut model is a running mean, and a finite depth costs nothing at prefill.

D.6 Sparse and quantized decode at matched error

Figure 10 places sparse and quantized decode on the final state of each generation at one byte model. Each method is emulated in exact arithmetic, and every byte it reads is counted, metadata included: Quest's per-page key bounds, the block scales and residuals of Faster Flash Decoding, KIVI's group scales and its recent tokens kept in BF16. Quest keeps the sink and last page and takes the top pages to a token budget, shared by the group or per head; Faster Flash Decoding keeps sub-blocks whose screened score reaches the pseudo-maximum of the first and last blocks less δ, and drops the others' mass. FoldAttention's points are its kernel's own counts. The same emulation gives the running-maximum gate of Section 6.2: each split's maximum is taken over its 64-key tiles up to the current one, and the kernel's refine and depth gates are applied against it with the same coarse logits.

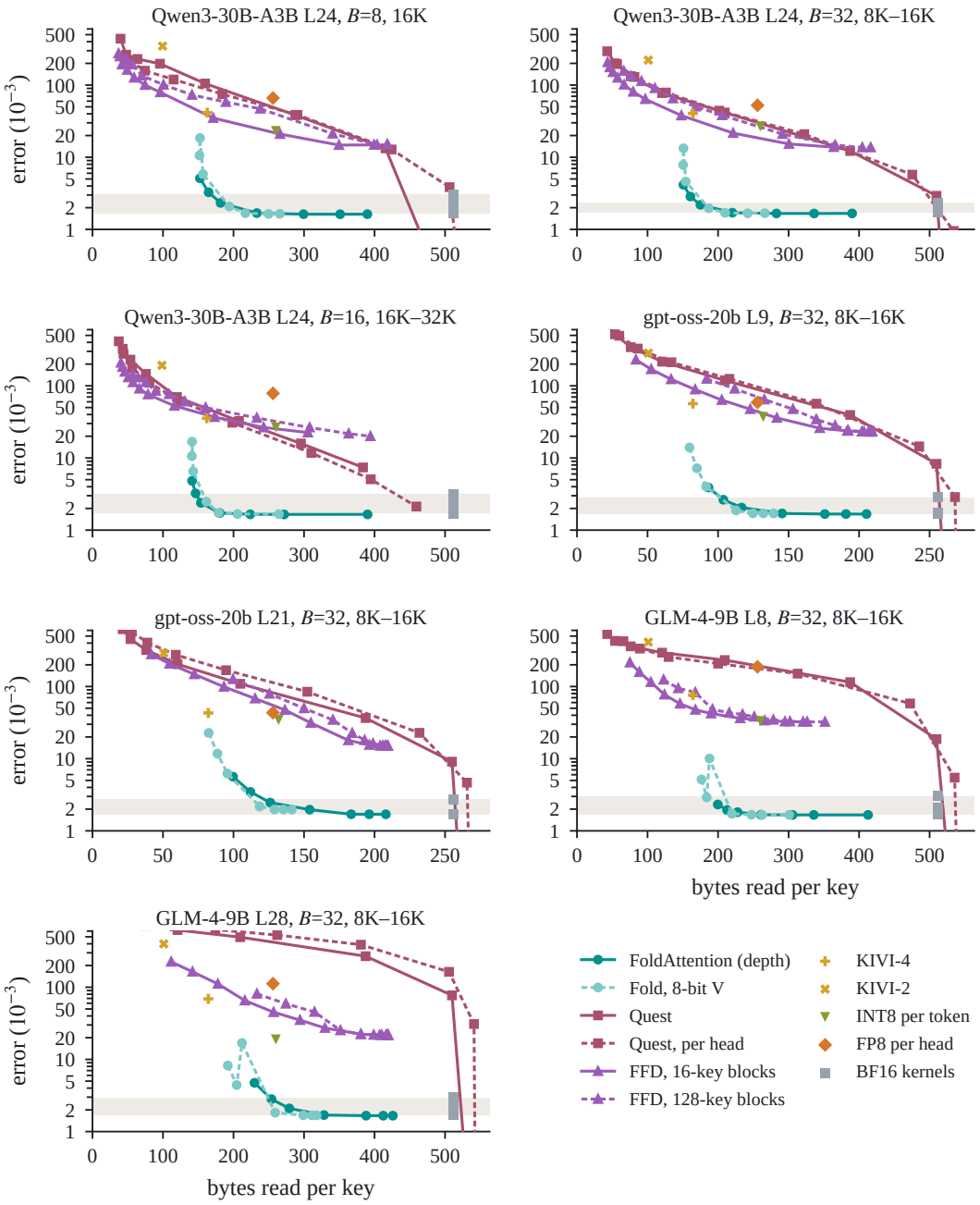

Figure 10: FP32-relative error against bytes read per key on the final state of each generation, every method emulated in exact arithmetic with its metadata counted. The shaded band spans the BF16 kernels' errors, which read $4D$ bytes per key. FoldAttention is a curve over depth T, from dense at the right; Quest and Faster Flash Decoding are curves over their budgets.

D.7 Cascades and drafts

The cascade takes its references from the flat decode's estimate, because the estimate's prepass sees only the level it runs on. Charging the flat decode's whole front to it, while the baselines' times exclude their own appends, gives 0.90–1.23× (1.07×) over the fastest kernel on cascades and 0.79–1.13× (0.98×) on trees. Table 5 gives the kernel-level speedups over each prefix-sharing kernel. FlashInfer's own paged decode is 2.3–7.9× faster than its cascade wrapper at these sizes. On draft verification (both models, 8 or 32 requests, a 4K or 16K cache), dense FoldAttention is 1.07–1.24×

faster than the fastest library for chains of two drafts, 0.99–1.17× for four, 0.58–0.84× for eight, and 0.43–0.74× for sixteen, and 0.71–1.09× and 0.72–1.14× against SGLang's tree verification for trees of eight and sixteen. The drafts share every byte the kernel reads, but each adds a column to every product, so the work grows with the number of drafts while the traffic does not.

Table 5: Cascade speedup of dense FoldAttention over each kernel: the range of the kernel's time over ours, with the geometric mean in parentheses. "Fastest" takes the fastest kernel of any kind in each cell. PAT has no gpt-oss-20b cells and FastTree no result on three cascades (Appendix C).

Set	FlashInfer cascade	vLLM cascade	PAT	FastTree	Fastest
Cascades (16)	3.93–8.88 (6.18)	1.08–1.76 (1.34)	1.13–1.48 (1.32)	1.23–1.95 (1.61)	1.04–1.42 (1.22)
Prefix trees (8)	3.38–8.88 (5.27)	1.48–2.88 (1.80)	0.90–1.29 (1.07)	1.02–1.71 (1.24)	0.90–1.33 (1.09)

E Additional backward results

Figures 11 and 12 give the grid FlashAttention-3 and DASH report: MHA and GQA with groups of eight, with and without a causal mask. Table 6 gives every model shape of Table 4.

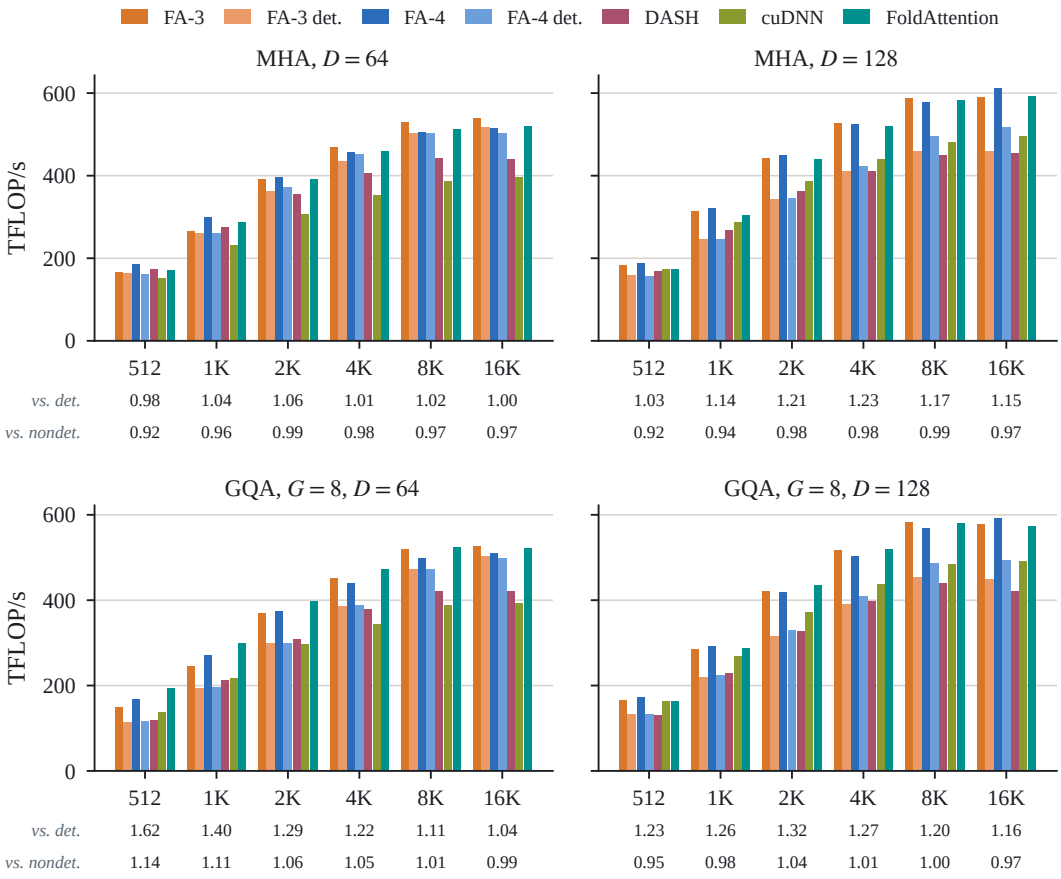

Figure 11: Backward throughput with a causal mask on the grid FlashAttention-3 and DASH report (16K tokens per batch, model width 2048), for MHA and for GQA with eight query heads per KV head. Hatched bars are deterministic modes. The table under each panel is FoldAttention's speedup over the fastest deterministic and the fastest nondeterministic kernel.

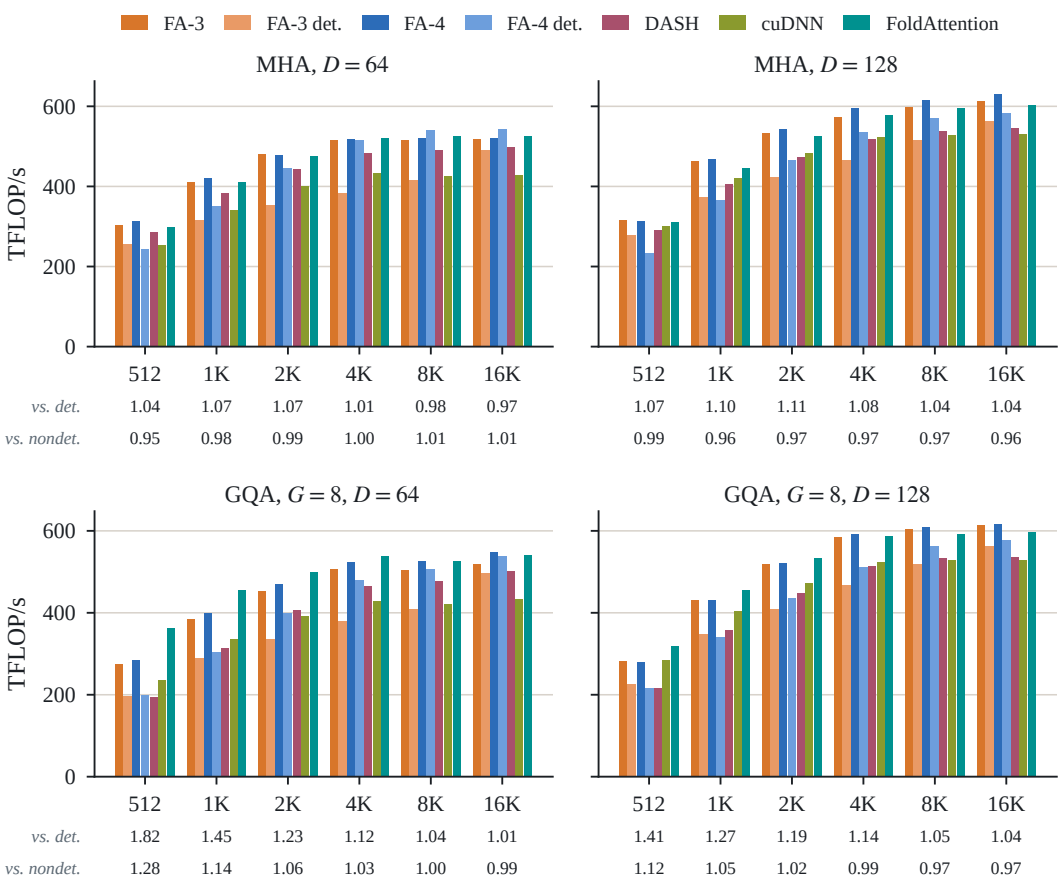

Figure 12: The grid of Figure 11 without a mask.

Table 6: Backward time of FoldAttention and each baseline's time over it. H/H_{KV} gives query and KV heads. Shapes marked † are below 16K tokens per batch and are excluded from the headline geometric mean.

B	H/H_{KV}	S	D	mask	ours (μs)	FA-3	FA-3 det.	FA-4	FA-4 det.	cuDNN	DASH
2	32/8	8192	128	causal	4942	1.03	1.29	1.04	1.19	1.21	1.37
4	32/8	4096	128	causal	2599	1.05	1.35	1.05	1.28	1.20	1.37
8	32/8	2048	128	causal	1548	1.10	1.38	1.08	1.34	1.18	1.39
16	32/8	1024	128	causal	1056	1.21	1.43	1.16	1.39	1.18	1.41
2	32/32	8192	128	causal	4865	1.00	1.25	0.99	1.14	1.19	1.30
2	32/8	8192	128	full	9026	1.03	1.17	1.00	1.07	1.16	1.16
2	64/8	8192	64	causal	5438	1.02	1.11	1.07	1.11	1.36	1.24
8	64/8	2048	64	causal	1660	1.12	1.39	1.09	1.37	1.38	1.32
1	8/2	8192	128	causal†	571	1.02	1.47	1.17	1.35	1.36	1.36
1	8/2	2048	128	causal†	83	1.14	1.38	1.08	1.28	1.27	1.58
1	16/2	4096	64	causal†	197	0.99	1.30	1.16	1.30	1.51	1.24

Table 7: Per-element gradient error against FP64 on attention operands captured from the training run of Section 6.4 (a Llama-3.2-1B-shaped model, $H/H_{KV} = 32/8$, $D = 64$, 8192 tokens), for FoldAttention and FlashAttention-3. ℓ_2 is the dQ error in 10^{-3}; p99 is the 99th percentile of per-element relative error. The binade columns give FoldAttention's median relative error over FlashAttention-3's among dQ elements that many binades below the largest of their request and KV head; the last columns cover elements 2^{-20} or more below it. dK and dV match FlashAttention-3's ℓ_2 error to three digits in every layer.

Step, layer	ℓ_2 Fold / FA-3	p99 Fold / FA-3	0 to -13	-14 to -19	share ≤ -20	median ≤ -20, Fold / FA-3
500, 0	2.99 / 2.99	1.45 / 1.52	1.00–1.03	1.07–1.12	2.7%	0.58 / 0.46
500, 7	5.01 / 5.01	12.5 / 19.0	1.00	1.00	6.3%	1.12 / 2.20
500, 15	5.01 / 5.01	12.0 / 14.8	1.00	1.00	5.7%	1.30 / 2.03
1999, 0	3.36 / 3.36	1.23 / 1.25	1.00–1.04	1.08–1.21	2.1%	0.40 / 0.31
1999, 7	5.67 / 5.67	3.17 / 3.27	1.00	1.00	0.9%	2.26 / 2.67
1999, 15	4.43 / 4.44	1.69 / 1.54	1.00	1.00–2.70	2.2%	0.46 / 0.0047

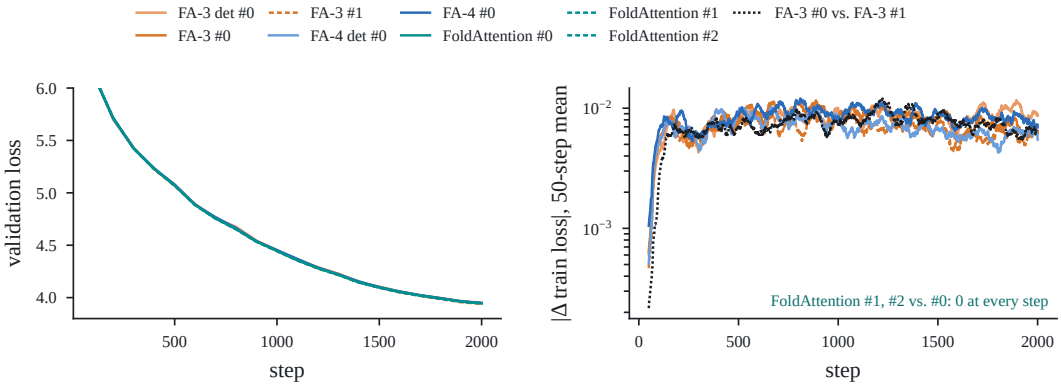

Figure 13: A 1B Llama trained from scratch on WikiText-103 for 2000 steps of 16K tokens with each library's attention, from the same initialization and data order. Left: validation loss. Right: each run's training loss against FoldAttention's first run, as a 50-step mean of the absolute difference. FoldAttention's second and third runs match its first at every step and have no point on the log scale; the dotted line is FlashAttention-3 against itself.

F Determinism

As DASH and TBIK do (Qiang et al., 2026; Zhang et al., 2025), Table 8 measures the largest change in any gradient element of one request when it is rerun, batched, or packed.

Table 8: Largest absolute difference in any dQ, dK, dV element of one request against its first result, over three dense shapes and two packed ones (0 means every bit matched). "Repeatable" counts backward benchmark shapes with identical bits over three calls; error is the worst FP32-relative ℓ_2 over dQ, dK, dV in 10^{-3}.

Kernel	Ten runs	Alone vs. batch of 4	Alone vs. packed	Repeatable	Worst error
FA-3	3.9×10^{-3}	2.0×10^{-3}	2.0×10^{-3}	1/73	2.53
FA-4	2.0×10^{-3}	4.9×10^{-4}	2.4×10^{-4}	3/141	2.53
cuDNN	4.9×10^{-4}	4.9×10^{-4}	–	11/136	3.02
FA-3 det.	0	0	0	73/73	2.53
FA-4 det.	0	0	0	141/141	2.53
DASH	0	0	–	68/68	2.52
FoldAttention	0	0	0	141/141	2.53

Table 9 measures decode: the same request run twice, and run alone instead of inside a ragged batch. A fixed number of keys per split also makes decode invariant to the split count; with the best such chunk per model, the generations' steps are 2% faster to 6% slower than at the default split.

Table 9: Largest absolute output difference for the first request of a ragged batch of 32 at 16K context, run twice, and run alone instead of in the batch. FoldAttention's default split is chosen from the batch (8 splits in the batch, 64 alone); a fixed split of 4 makes it batch invariant. Changing the fixed split from 4 to 8 changes the combine's grouping and the bits. cuDNN 9.26's paged decode is not repeatable on the $D = 128$ batch.

	Qwen3-30B, $D = 128$		gpt-oss-20b, $D = 64$	
Kernel	repeat	alone vs. batch	repeat	alone vs. batch
FA-3	0	4.9×10^{-4}	0	1.6×10^{-2}
FA-4	0	0	0	0
FlashInfer	0	2.0×10^{-3}	0	1.6×10^{-2}
XQA	0	2.0×10^{-3}	0	3.1×10^{-2}
cuDNN	2.4×10^{-4}	9.8×10^{-4}	0	1.6×10^{-2}
FoldAttention, default split	0	1.5×10^{-8}	0	7.6×10^{-6}
FoldAttention, fixed split	0	0	0	0