SHIFT-ACCUMULATE ATTENTION: MULTIPLIER-FREE QUERY–KEY PRODUCTS FOR TRANSFORMER DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Power-of-two quantisation turns a multiplication into a bit shift, but it has so far been applied only to the post-softmax attention–value product. We reformulate the earlier and larger product, $\mathbf{S} = \mathbf{Q}\mathbf{K}^T$, by quantising the key cache to a signed power-of-two fixed-point value, so that every scalar multiplication in the score computation becomes a sign flip, a shift and an integer accumulation. We call the resulting datapath ShiftAtten. A fixed-point head-room $F \geq e_{\max}$ makes every shift a non-negative left shift, which renders the accumulation exact with respect to the quantised operands; a mantissa field then trades extra shift-adds for accuracy across a 4-to-8-bit family, and a shift-exact online softmax whose running maximum lives in an integer $\log_2$ domain extends the same treatment to $\mathbf{AV}$. Fused CUDA kernels for a 1.1 B Llama decoder on an RTX 4090 reach $4.60\times$ the throughput of FP16 scaled dot-product attention at T=32k with a $2.5\times$ smaller KV cache. We separate the two sources of that gain: $2.53\times$ is the narrower representation and $1.81\times$ is kernel engineering. On zero-shot reasoning the 8-bit shift code is a drop-in (61.61% against FP16's 61.51%), and at four bits it retains nearly three times as much accuracy as uniform INT4 (-0.94 against -2.68 points) despite worse perplexity and tensor-level error, a disagreement between proxies and task accuracy that we report as measured. A cost model calibrated by disassembly locates the remaining bottleneck in instruction issue rather than memory: current GPUs expose a 4-way integer multiply–accumulate but no shift equivalent, and PTX's scalar shift-accumulate is emulated. Given that missing primitive, a packed shift-dot-accumulate (DS4A), the identical kernel would reach $6.51\times$ over FP16, beyond the $4.94\times$ an optimised INT8 MAC kernel achieves today, because the format moves 101 bytes per cached token against 133.

1 INTRODUCTION

Scaled dot-product attention Vaswani et al. (2017) contains two dense matrix products,

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T, \qquad \mathbf{O} = \mathbf{A}\mathbf{V}, \quad \mathbf{A} = \mathrm{softmax}\big(\mathbf{S}/\sqrt{d}\big), \tag{1}$$

both conventionally evaluated on multiply–accumulate (MAC) hardware. If an operand is constrained to a signed power of two, $\sigma 2^{-e}$ with $\sigma \in \{-1, +1\}$, the multiplication collapses to a shift, an optional negation and an addition, replacing a multiplier array with a barrel shifter and an adder, which is the central efficiency argument of the multiplication-free network literature Chen et al. (2020); You et al. (2020).

FQ-ViT Lin et al. (2022) applies this to the post-softmax matrix, logarithmically quantising $\mathbf{A}$ and evaluating $\mathbf{AV}$ with bit shifts. I-ViT Li & Gu (2023) performs integer-only inference with dyadic arithmetic, ShiftAddViT You et al. (2023) reparameterises attention with binary and additive kernels, and P^2-ViT Shi et al. (2024) develops power-of-two post-training quantisation with hardware co-design. What no prior work does is represent one operand of $\mathbf{Q}\mathbf{K}^T$ itself as a signed power-of-two fixed-point code, so that the score computation becomes shift-and-accumulate in ordinary scaled dot-product attention.

That is the larger of the two opportunities in a decoder, for a reason specific to autoregressive generation. Prefill is compute-bound and well served by tensor cores, but each subsequent token reduces

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.

a single query vector against the entire key cache, so the step is bound by how fast that cache can be read rather than by arithmetic throughput. The cache grows linearly in both sequence length and batch size, and at long context it dominates the memory traffic of the whole model. Its bit-width therefore sets the decode rate, which is why quantising it has become a standard lever Liu et al. (2024); Hooper et al. (2024) and why eviction schemes attack the same quantity from the other side Zhang et al. (2023); Liu et al. (2023). What those methods do not change is the arithmetic: a narrower bit pattern is still unpacked and fed to a multiplier.

Constraining the representation to powers of two changes this trade-off, offering computational benefits beyond the reduction in storage. A multiplier array is the expensive element of a MAC datapath in both area and energy, and removing it is the premise of an entire line of accelerator work. The difficulty is that a logarithmic representation is a poor fit for the places where quantisation is usually applied., and that shifting is only free if the shifts can be made exact: a right shift discards low-order bits before they are accumulated, which is precisely what destroys accuracy in naive fixed-point attention. We show that a head-room condition removes the problem entirely, making the reduction exact with respect to the quantised operands, and that the same treatment extends through the softmax to the second attention product. We call the resulting datapath ShiftAtten.

The remaining question is whether multiplier-free arithmetic translates into a practical advantage on GPUs optimised for multiplication. NVIDIA provides `__dp4a`, a four-way packed INT8 dot-product–accumulate intrinsic, whereas PTX documents `vshl`/`vshr` as scalar shift operations and provides no corresponding packed shift primitive NVIDIA Corporation (2026a;b). Our disassembly confirms that shift–accumulate therefore requires multiple scalar instructions, motivating the packed DS4A primitive proposed in Sec. 4.

Contributions. (i) ShiftAtten, a shift/sign/accumulate formulation of $\mathbf{QK}^T$ that is exact with respect to the quantised operands (Sec. 3). (ii) A mantissa-extended power-of-two family PoT-Mk spanning 4 to 8 stored bits, in which PoT-M4 matches an INT8 key cache in storage and end-to-end perplexity while remaining multiplier-free. (iii) Fused CUDA decode kernels and a KV cache that stores only quantised codes, giving up to $4.60\times$ FP16 attention throughput. (iv) An ISA analysis showing that PTX's scalar shift-accumulate is emulated on current hardware while no packed shift-dot-accumulate exists, and a cost model, calibrated on disassembled kernels, quantifying what such an instruction, which we call DS4A, would be worth (Sec. 4).

2 Related work

Logarithmic data representations date to Miyashita et al. (2016), and additive powers-of-two Li et al. (2020) densify the codebook with a few extra shift-adds; our PoT-Mk family is a structured special case whose second term sits a fixed binade below the first, so its offset is known at compile time. For language models, KIVI Liu et al. (2024) quantises the KV cache to 2 bits with asymmetric uniform codes, per channel for keys and per token for values, while KVQuant Hooper et al. (2024) uses a non-uniform, sensitivity-derived codebook with a dense-and-sparse split that isolates outliers. Our key cache is likewise non-uniform (logarithmically spaced), so the distinction is not uniformity but structure. An arbitrary non-uniform codebook dequantises to levels that still need a multiply, whereas constraining the levels to powers of two lets a shift consume them directly. That constraint is what makes the score product multiplier-free. Orthogonal to quantising the cache, token-eviction methods such as H_2O Zhang et al. (2023) and Scissorhands Liu et al. (2023) shrink it by discarding entries, and Atom Zhao et al. (2024) combines low-bit weights and activations for serving; these compose with a narrower code rather than competing with it. Weight-only schemes such as GPTQ Frantar et al. (2023) and AWQ Lin et al. (2024) address a different tensor again. PTQ studies for ViTs Liu et al. (2021); Yuan et al. (2022) established that preserving the ranking of attention scores matters more than minimising mean-squared error, which is why we report top-k overlap alongside ε_S. SmoothQuant Xiao et al. (2023) and QuaRot Ashkboos et al. (2024) supply exactly score-preserving outlier transformations, which we use as a normalisation axis. FlashAttention Dao et al. (2022); Dao (2023) and online softmax Milakov & Gimelshein (2018) define the kernel structure any new attention datapath must match.

Attention accelerators. The architecture community has attacked the same bottleneck in silicon rather than in software. A^3 Ham et al. (2020) and ELSA Ham et al. (2021) approximate the score product to skip work, SpAtten Wang et al. (2021) and Sanger Lu et al. (2021) prune tokens and heads

Table 1: Positioning against the closest prior work. "Levels" is the number of distinct values the attention operand can take. Only ShiftAtten makes the dynamic $\mathbf{K}$ activation itself multi-level power-of-two and evaluates both attention products with shifts.

Method	Power-of-two applied to	$\mathbf{QK}^T$	$\mathbf{AV}$	levels
I-ViT Li & Gu (2023)	requant. scale, nonlin.	MAC	MAC	255
P^2-ViT Shi et al. (2024)	quantisation scale	MAC	MAC	255
FQ-ViT Lin et al. (2022)	post-softmax $\mathbf{A}$	MAC	shift	255
ShiftAddViT You et al. (2023)	MLP weights; attn. operand	add	add	2
ShiftAtten (ours)	**dynamic $\mathbf{K}$ activations**	**shift**	**shift**	**15–225**

on reconfigurable hardware, and FlightLLM Zeng et al. (2024) maps a complete LLM inference flow onto an FPGA. Closest in spirit to the ShiftAtten format are the datatype co-designs: ANT Guo et al. (2022) adapts the numerical type per tensor, OliVe Guo et al. (2023) pairs outliers with victims so a hardware-friendly encoding survives them, and Tender Lee et al. (2024) requantises at runtime. DeepShift Elhoushi et al. (2021) replaces multiplication with shifts in convolutional networks. What none of these do is make the shift the primitive of the attention score product itself, which is what determines whether a packed shift instruction would be worth adding (Sec. 4).

Table 1 separates the three closest works along the axis that matters. I-ViT and P^2-ViT both use shifts, but for scaling: I-ViT approximates the requantisation factor by a dyadic number and its nonlinearities by `Shiftmax`/`ShiftGELU`, while P^2-ViT constrains the quantisation scale to a power of two; in both, $\mathbf{QK}^T$ remains an ordinary integer MAC. FQ-ViT is the direct precedent for the second product only. ShiftAddViT is closest, since it does remove multipliers from the attention MatMuls, but it does so by binarising an operand, so the attention operand carries two levels. Our method is the multi-level generalisation of that special case: binary attention is signed-PoT attention restricted to $e{=}0$, and widening the exponent and mantissa fields recovers 15 to 225 levels while keeping the datapath multiplier-free. That is the whole of our novelty claim: we do not claim power-of-two quantisation for Transformers, nor shift arithmetic for Transformers, both of which are established.

3 Shift-Accumulate Attention

3.1 Signed power-of-two keys

Let $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{T\times d}$ be post-RoPE queries and keys of one head. The query is quantised to INT8 with a per-vector step Δ_i, and the key to a signed power-of-two code with a per-key-vector scale s_j:

$$Q_{im} \approx \Delta_i \hat{q}_{im}, \qquad \hat{K}_{jm} = s_j\, \sigma_{jm} 2^{-e_{jm}}, \;\; e_{jm} \in \{0, \dots, E\}, \tag{2}$$

with the code $e = E{+}1$ reserved for an exact zero. A b-bit field spends one bit on σ and $b-1$ on the exponent, so $b{=}4$ gives $E{=}6$.

3.2 The shift-accumulate score product

Substituting (2) into the score and introducing a common fixed-point head-room F with shift index $r_{jm} = F - e_{jm}$ gives

$$\boxed{\hat{S}_{ij} = \Delta_i s_j 2^{-F} \sum_{m=1}^{d} \sigma_{jm} \big(\hat{q}_{im} \ll r_{jm}\big)} \tag{3}$$

The inner sum is a pure integer shift/sign/accumulate (SAC) reduction: one barrel shift, one conditional negation and one add per term. The two floating point factors and the common 2^{-F} are applied once per (i, j), after the accumulator is formed, not per term.

3.3 Exactness

Because $\mathbf{K}$ is normalised by s_j before quantisation, all $e_{jm} \geq 0$, so

$$F \;\geq\; E \quad\Longrightarrow\quad r_{jm} \geq 0 \tag{4}$$

and every shift inside the accumulation is a non-negative left shift. Equation (3) is then evaluated exactly in integer arithmetic: no truncation is introduced, and the only error is the operand quantisation error of Equation (2). This is stronger than the usual fixed-point argument, which merely states that head-room reduces truncation. With d=64, F=7 the accumulator is bounded by $d \cdot 127 \cdot 2^F = 1.04 \times 10^6$, well inside INT32.

Note that Equation (4) depends on the code, not on the data: keys are normalised by their per-vector maximum before quantisation, so $e_{jm} \geq 0$ holds by construction whatever the outlier structure. Exactness therefore cannot be invalidated by a shift in activation statistics, which is relevant when moving to larger models (Sec. 6.1). The mantissa family of Sec. 3.4 decodes terms as small as $F - e_{jm} - k$, so preserving exactness there requires the stronger

$$F \;\geq\; E + k, \tag{5}$$

which our configurations satisfy with equality: F=7 for PoT-M1 (k=1) and F=10 for PoT-M4 (k=4).

3.4 Mantissa extension

The relative error of nearest-in-$\log_2$ quantisation of $u = 2^{-t}$ is $2^{\,t-\lfloor t \rceil} - 1 \in [-0.293, 0.414]$, with RMS $\big(\int_{-1/2}^{1/2} (2^x - 1)^2 dx\big)^{1/2} \approx 0.203$, independent of the number of exponents. Extra exponent bits therefore buy nothing beyond covering the dynamic range; only 5.0% of key entries underflow a three-bit exponent. Accuracy must instead come from subdividing the binade. A k-bit mantissa field encodes

$$\hat{K}^{(k)}_{jm} = s_j \, \sigma_{jm} \Big(1 + \tfrac{\mu_{jm}}{2^k}\Big) 2^{-e_{jm}}, \quad \mu_{jm} \in \{0, \ldots, 2^k - 1\}, \tag{6}$$

which decodes to one extra shift-and-add per set bit of μ_{jm}: a mean cost of $1 + k/2$ shifts. With three exponent bits the family spans 4 bits (PoT-4, one shift) to 8 bits (PoT-M4, up to five shifts), the latter being exactly one byte per element and hence iso-storage with an INT8 key cache.

3.5 Shift-exact softmax and $\mathbf{AV}$

Write the scaled score of (3) in the base-2 domain as

$$z_{ij} = \big(\hat{S}_{ij}/\sqrt{d}\big) \log_2 e, \tag{7}$$

where $\hat{S}_{ij}$ is the dequantised accumulator of (3); we write z rather than a lower-case s to avoid collision with the quantisation scale s_j of (2). A streaming softmax rescales its accumulators by $e^{-\delta}$ whenever the running maximum grows. Keeping that maximum as an integer, $M = \lceil \max_j z_{ij} \rceil$, makes every correction an exact shift:

$$k_j = \mathrm{round}(M - z_{ij}) \geq 0, \qquad \tilde{A}_{ij} = 2^{-k_j}, \tag{8}$$

so an increase ΔM rescales by $2^{-\Delta M}$. With $\mathbf{V}$ in INT8 with a power-of-two per-token scale 2^{-g_j}, numerator and softmax denominator are accumulated in the same shifted integer domain,

$$a_i = \sum_{j : k_j \leq k_{\max}} \big(\hat{v}_j \ll \rho_j\big), \quad \ell_i = \sum_{j : k_j \leq k_{\max}} 2^{\,N - k_j}, \quad \rho_j = N - k_j + g_{\mathrm{ref}} - g_j, \tag{9}$$

and the common 2^{-N} cancels between them, leaving one division:

$$\mathbf{O}_i = 2^{-g_{\mathrm{ref}}} \, a_i / \ell_i. \tag{10}$$

Since ℓ_i is the integer form of $\sum_j \tilde{A}_{ij}$, (10) is the normalised output of (1), not an unnormalised accumulator.

The shift index ρ_j is bounded on both sides. Choosing $g_{\mathrm{ref}} = \min_j g_j$ gives $\rho_j \leq N$, which caps the accumulator width; a left shift additionally needs $\rho_j \geq 0$, i.e.

$$N \;\geq\; k_{\max} + \max_j g_j - g_{\mathrm{ref}}. \tag{11}$$

These two requirements conflict. With $|\hat{v}| \leq 127$ and a 128-key tile, INT32 forces $N \leq \lfloor \log_2(2^{31}/(127 \cdot 128)) \rfloor = 17$, while the measured spread $\max_j g_j - g_{\mathrm{ref}} = 6$ on our tensors makes (11) demand $N \geq 21$: no N satisfies both. We take N=16 and let the kernel fall back to a truncating right shift when $\rho_j < 0$. This is benign because the affected terms are those with the largest k_j, i.e. the smallest attention weights: they are 5.4% of retained terms but carry 0.06% of the attention mass.

Table 2: Native instruction lowering on `sm_89` for shift–accumulation implementations. The PTX scalar shift–accumulate is emulated, while plain C is already near optimal; only integer multiplication provides a packed four-way form. The `__dp4a` row covers 256 terms because each `IDP` performs four MACs, giving a cost below one instruction per term, whereas all other rows cover 64 terms.

Primitive	Form	ALU ops	terms	per term
`__dp4a`	$\sum_{i<4} a_i b_i + c$	64	256	0.25
plain C	$(a \ll b) + c$	103	64	1.61
`shl.b32`	$(a \ll b) + c$	103	64	1.61
`vshl.wrap.add`	$(a \ll b) + c$	295	64	4.61
`vshl.clamp.add`	$(a \ll b) + c$	423	64	6.61

3.6 Kernel and cache

Decoding is the target: a single query vector is reduced against the whole cache, so bit-width translates directly into time. The kernel launches (split, H, B) blocks of 128 threads, quantises its query in registers, and processes the cache in tiles of 128 keys. Phase 1 evaluates (3) with vectorised 16-byte code loads and a branch-free inner loop, phase 2 runs the block-wide online softmax (8), phase 3 stages the INT8 value tile in shared memory and evaluates (9). The cache stores only quantised data: for PoT-4 that is 101 bytes per cached token per head (32 nibble codes plus a 4-byte key scale, and 64 INT8 values plus a 1-byte exponent) against 256 for an FP16 K/V pair. A bit-exact PyTorch reference reproduces the kernel output to 2×10^{-4}.

4 Cost Model and Shift-Native Projection

Let R be the bytes a datapath requests per decode step, Θ_{mem} the bandwidth the kernel structure achieves, N the integer instructions it issues and Θ_{alu} the issue rate. To first order

$$T \;\approx\; \max\Big(\frac{R}{\Theta_{\text{mem}}},\; \frac{N}{\Theta_{\text{alu}}}\Big). \tag{12}$$

Disassembling the ShiftAtten kernels for `sm_89` shows which term binds. A 4-way integer dot product (`IDP`, exposed as `__dp4a`) performs four INT8 MACs in one instruction, i.e. 0.25 instructions per term. The shift datapath has no equivalent: because every element carries its own shift amount, no SIMD-within-a-register instruction can apply four different shifts to four bytes of one word. Each term therefore costs a separate extract, shift, mask and add, and the whole kernel issues 1011 integer ALU instructions against 409 for a MAC kernel of identical structure. Holding all d query values unpacked also costs registers, capping occupancy at 50% against 67%. The shift kernel is therefore issue-bound, not bandwidth-bound: the second term of (12) dominates.

4.1 The available instruction set

A scalar shift-accumulate is not missing from the ISA: PTX has exposed `vshl`/`vshr` with an optional secondary `.add` since `sm_20`, giving exactly $d = (a \ll b) + c$. PTX is a virtual ISA, however, so whether this is a hardware primitive depends on what `ptxas` emits. Table 2 settles it by disassembly.

The scalar path is emulated on this architecture. The clamping form costs 6.61 integer ALU operations per term against 1.61 for an ordinary C shift-add, a factor of 4.1; the wrap form costs 2.9. Almost all of that overhead is the clamp/wrap semantics on the shift amount (`PRMT`, `ISETP`, `SEL`), the secondary `.add` itself being free. Plain C meanwhile lowers to `SHF` plus `IADD3`, whose three-input form absorbs pairs of additions, so a bare shift-accumulate already costs 1.61 operations per term. Inline PTX therefore buys nothing, and ShiftAtten does not use it.

This locates the gap. What is missing is not a scalar shift-accumulate, which the compiler already generates near-optimally, but a packed one: the PTX SIMD video set offers `vadd4` and `vsub4` but lists `vshl`/`vshr` only in scalar form. The comparison therefore has three available levels, FP16 on tensor cores, INT8 on `IDP`, and power-of-two on scalar shift-add, with no packed shift equivalent.

Table 3: Instruction-aware projection under DS4A at B=32, T=32768, evaluating (14) with both rates calibrated on our measured kernels. Every code remains memory-bound even after the mantissa is charged its extra issues: the memory term exceeds the ALU term in every row.

Code	bytes /token	DS4A per 4 terms	memory (μs)	ALU (μs)	projected speed-up
PoT-4	101	1.0	1559	59	6.51×
PoT-M1	109	1.5	1683	89	6.03×
PoT-M4	133	3.0	2053	178	4.94×
INT8 MAC	133	1.0	2053	59	4.94×

4.2 A PROPOSED INSTRUCTION: DS4A

__dp4a computes $d = \sum_{i<4} a_i b_i + c$ in one instruction. Its shift analogue, which we call DS4A, would compute

$$\mathrm{DS4A}(Q, E, \Sigma, c, F) = c + \sum_{i=0}^{3} \sigma_i \left(q_i \ll (F - e_i)\right) [\, e_i \le E \,], \tag{13}$$

with the head-room F an immediate operand, so the shift index matches r_{jm} of (3) without storing it in the cache, and the Iverson bracket realising the reserved zero code of (2): a sign bit cannot nullify a term, so a lane must suppress it on the maximal exponent. That is one comparator per lane, which the software kernel already pays as a mask.

Each lane still performs a single shift, so a PoT-Mk code needs $1 + k/2$ DS4A issues per four terms rather than one, and the projection must charge that rather than assume instruction parity. Evaluating (12) with both rates calibrated on our kernels, $\Theta_{\mathrm{mem}} = 2.17$ TB/s from the memory-bound INT8 kernel and $\Theta_{\mathrm{alu}} = 9.05$ Tinstr/s from the issue-bound PoT-4 kernel,

$$T_{\mathrm{proj}} = \max\Big(\frac{R}{\Theta_{\mathrm{mem}}},\ \frac{(1 + k/2)\, T_{\mathrm{terms}}/4}{\Theta_{\mathrm{alu}}}\Big). \tag{14}$$

Table 3 shows that the correction does not change the projections. Even PoT-M4, at three issues per four terms, spends 178 μs of issue against 2053 μs of memory, and would become issue-bound only if DS4A issued below 8.7% of the integer throughput we measure. That PoT-M4 and INT8 coincide is the model's consistency check: they are iso-storage, and under DS4A both are memory-bound.

The unit is modest, and cheaper than the one it sits beside: four lanes of shift, conditional negate and adder tree replace four 8×8 multiplier arrays, and because the exponent field is three bits the shifter is a three-stage multiplexer rather than a full barrel shifter. The shift-based network literature Chen et al. (2020); You et al. (2020); Elhoushi et al. (2021) rests on exactly this asymmetry, and the accelerator designs of Sec. 2 Guo et al. (2022; 2023); Zeng et al. (2024) show that a custom numerical type earns its silicon only when the datapath consuming it is correspondingly cheaper. The operand width is a second, less obvious advantage: IDP consumes two 32-bit words, whereas DS4A would consume one word of INT8 queries but only

$$4 \times (1 + 3) = 16 \text{ bits} \tag{15}$$

of key data, halving the register and cache pressure of the streaming operand at the instruction boundary. Equation (14) remains an upper bound under the stated assumptions rather than a hardware simulation.

5 EXPERIMENTAL SETUP

One NVIDIA RTX 4090 (24 GB, sm_89), CUDA 12, PyTorch 2.8, Transformers 4.56. Model: TinyLlama-1.1B-Chat Zhang et al. (2024) (22 layers, 32 heads, 4 KV heads, d=64). Perplexity on WikiText-103 Merity et al. (2017) over 2048-token windows. Quantiser errors are measured on post-RoPE tensors captured from five layers: ε_S is the relative score error, D_{KL} the mean attention-distribution divergence, and top-8 the retained ranking overlap. Kernel timings are the best of three runs after warm-up, against scaled_dot_product_attention on an unquantised FP16 cache.

Table 4: Key representations on real post-RoPE attention tensors, five layers per model, identical protocol at both scales. ε_S is the relative score error and top-8 the retained ranking overlap. The ordering of the codes is preserved from 1.1 B to 8 B and the errors do not grow with scale.

		TinyLlama-1.1B (d=64)			Llama-3.1-8B (d=128)	
Key code	Format	ε_S	D_{KL}	top-8	ε_S	top-8
INT8	8 b, 1 mul	0.0042	0.00012	0.989	0.0048	0.985
INT4	4 b, 1 mul	0.0763	0.03754	0.845	0.0855	0.798
PoT-4	4 b, 1 shift	0.1302	0.06881	0.777	0.1223	0.743
PoT-M1	5 b, 2 shifts	0.0626	0.01811	0.869	0.0596	0.846
PoT-M2	6 b, 3 shifts	0.0314	0.00478	0.925	0.0294	0.917
PoT-M4	8 b, 5 shifts	0.0087	0.00041	0.978	0.0087	0.973

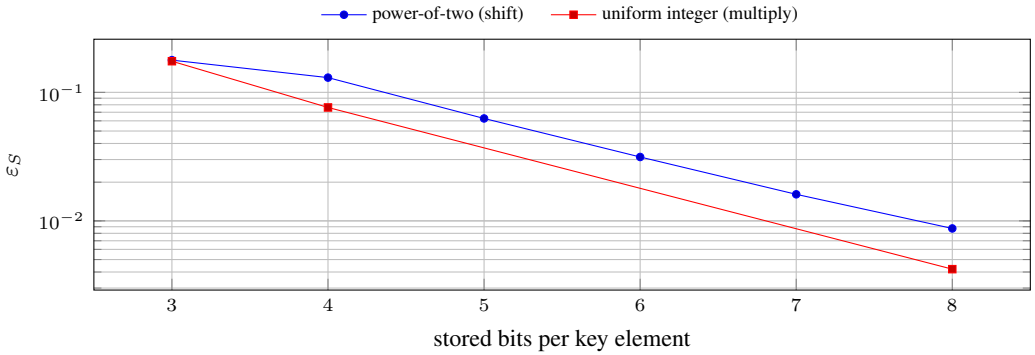


Figure 1: Accuracy per stored bit. The power-of-two family is flat until a mantissa field is added; each mantissa bit roughly halves ε_S at the cost of one shift–add per term.

6 Results

6.1 Accuracy of the key codes

Table 4 shows that increasing the exponent width beyond three bits provides no measurable benefit ($\varepsilon_S = 0.1302$), whereas each mantissa bit roughly halves the score error, $0.1302 \to 0.0626 \to 0.0314$, at one additional shift–add per term. At 8 bits, PoT-M4 reaches 0.0087 versus 0.0042 for INT8; although less accurate at matched width, the gap is small enough for indistinguishable end-to-end perplexity (Sec. 6.4), and Hadamard rotation Ashkboos et al. (2024) further reduces the error to 0.0059. Thus, comparable accuracy is obtained while removing the multiplier.

The right-hand columns of Table 4 repeat the experiment on Llama-3.1-8B Grattafiori et al. (2024) (d=128). The exactness condition in (4) is scale-independent because per-vector normalisation guarantees $e_{jm} \geq 0$, while the empirical quantisation error also does not deteriorate: the representation ordering is unchanged, ε_S is slightly lower for each PoT configuration, and three-bit exponent underflow changes only from 5.03% to 5.69%. Additional quantiser choices are reported in Appendix A.

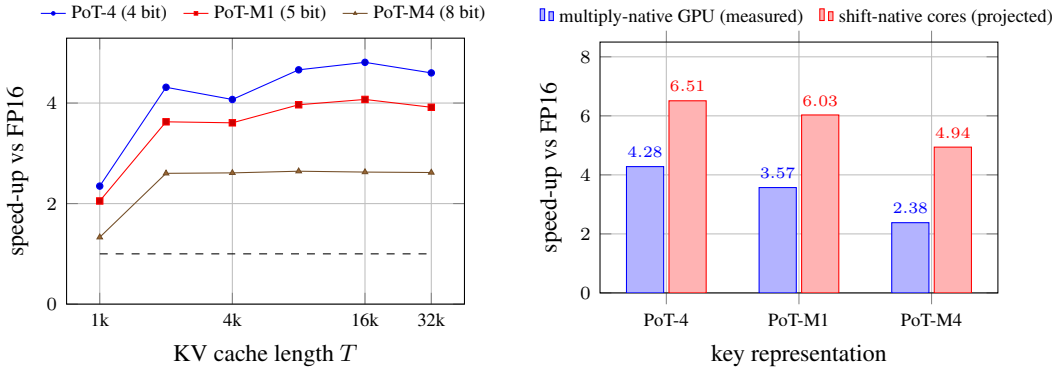
6.2 Kernel throughput against FP16

Fig. 2a and Table 5 give the measured picture. At short contexts the fused kernel is launch-bound and at parity with FP16; from $T{\approx}$4k the ordering is stable, and at B=8, T=32k PoT-4 reaches $4.60\times$ FP16 SDPA and PoT-M1 $3.92\times$.

That factor is not all representation, and it is worth separating. PoT-4 moves 101 bytes per cached token against 256 for FP16, which caps a purely bandwidth-driven gain at $2.53\times$. The remaining $1.81\times$ is kernel engineering: PyTorch SDPA sustains 0.77 TB/s of requested traffic at this shape while the ShiftAtten kernel sustains 1.40 TB/s, so the baseline is not achieving its own memory bound. The product 2.53×1.81 recovers the measured $4.60\times$ exactly. Only the first factor follows

Table 5: Decode-attention speed-up over FP16 SDPA. Left: measured on today's multiply-native GPU, including the INT8 `__dp4a` kernel built on the same tiling. Right: projected under (14) for shift-native cores.

workload		FP16	measured speed-up, multiply-native GPU				projected, DS4A	
B	T	(μs)	PoT-4	PoT-M1	PoT-M4	INT8 MAC	PoT-4	PoT-M1
8	4096	357	**4.07**	3.61	2.61	5.05	**6.64**	6.16
8	8192	711	**4.66**	3.97	2.64	5.36	**7.06**	6.54
8	16384	1404	**4.81**	4.07	2.63	5.29	**6.97**	6.46
8	32768	2787	**4.60**	3.92	2.62	5.29	**6.96**	6.45
32	8192	2579	**4.33**	3.64	2.43	4.89	**6.43**	5.96
32	16384	5102	**4.28**	3.68	2.36	4.95	**6.52**	6.04
32	32768	10152	**4.28**	3.57	2.38	4.94	**6.51**	6.03



(a) Measured speed-up versus KV-cache length at B=8; the gain appears once the cache exceeds the cache hierarchy.

(b) Measured and projected speed-up at B=32, T=32768, assuming a 4-way shift–accumulate with `__dp4a` throughput.

Figure 2: Decode-attention speed-up over FP16 SDPA.

from the format; the second would also be available to a well-written INT8 kernel, which is why Table 5 carries the INT8 column: on current hardware it reaches $4.94\times$ and remains the strongest baseline.

6.3 Projected throughput under DS4A

Fig. 2b and the right half of Table 5 apply (14). With a 4-way shift-accumulate instruction the same kernel would reach $6.51\times$ FP16 at B=32, T=32k, a further $1.52\times$ over what we measure, and PoT-M1 $6.03\times$ at 5 stored bits. The gap between the two halves of the table is the issue-rate term of (12), which Sec. 4.1 attributes to the instruction set rather than to the arithmetic.

Two kernel-level decisions that changed the measured speed-up by more than a factor of two, and the reason the shift-based $\mathbf{AV}$ of (9) is offered as an option rather than the default, are set out in Appendices B and C.

6.4 End-to-end quality and throughput

Table 6 evaluates the complete decoder with a cache containing only quantised representations. At 8 bits, PoT-M4 with Hadamard rotation Ashkboos et al. (2024) closely matches FP16 in perplexity (7.896 versus 7.881) and zero-shot accuracy (61.61% versus 61.51%). At 5 bits, PoT-M1 reduces the cache from 91 to 39 MB, with its zero-shot loss reduced from -0.71 to -0.17 points by rotation.

At 4 bits, PoT-4 loses only -0.94 zero-shot points versus -2.68 for INT4 despite worse score error, top-8 overlap and perplexity. We report this inversion without claiming a mechanism from

Table 6: End-to-end with FP16 weights. Perplexity over 2048-token windows; zero-shot accuracy on the full validation sets (HellaSwag subsampled to 2000); throughput and KV cache at a 4096-token prompt, batch B=1. Bold marks the best quantised entry in each column, FP16 being the reference. INT4 KV is a reference point for the code family; no dedicated INT4 decode kernel was written, so its throughput columns are omitted.

			zero-shot accuracy (%)					
Attention	Bits	PPL	HellaS.	PIQA	Wino.	ARC-e	tok/s	KV (MB)
FP16 (reference)	16	7.881	55.80	74.59	60.38	55.25	203	91
INT8 KV (MAC)	8	7.902	55.75	74.54	60.30	55.15	**187**	47
INT4 KV (uniform)	4	8.386	53.10	72.36	58.09	51.75	–	–
PoT-4	4	9.303	54.60	73.34	60.06	54.25	183	**36**
PoT-4 + Hadamard	4	8.507	54.25	73.61	58.48	53.45	170	**36**
PoT-M1	5	8.173	54.70	74.59	59.43	54.45	182	39
PoT-M1 + Hadamard	5	8.029	**56.20**	**74.97**	58.72	**55.45**	168	39
PoT-M4	8	7.904	55.80	74.70	**60.54**	55.40	182	47
PoT-M4 + Hadamard	8	**7.896**	55.80	74.86	59.59	54.85	170	47

one model; it shows that tensor-level proxies alone need not predict downstream accuracy across logarithmic and uniform representations.

At B=1, PoT remains slower end-to-end despite the $> 4\times$ attention speed-up in Table 5, because linear-layer weight traffic dominates. With batching, KV traffic grows while weight traffic does not: at B=64, PoT-4 reaches 5131 tokens/s versus 4198 for FP16, while peak memory falls from 13.8 to 11.0 GB.

7 Discussion and Limitations

The scale study above is tensor-level only: Llama-3.1-8B uses d=128 and the ShiftAtten kernel is compiled for d=64, so no end-to-end 8 B throughput or perplexity is reported, and that d=64 restriction is a limitation of the present implementation rather than of the method. Sweep conclusions are drawn from a handful of layers per model. The projection (14) is an upper bound under the assumptions stated in Sec. 4. Prefill is not fused, so prefill latency reflects the reference implementation. The zero-shot suite of Sec. 6.4 covers single-forward-pass discrimination; it does not probe multi-step generation.

8 Conclusion

Quantising the key cache to a signed power-of-two code turns the query–key product into an exact integer shift/sign/accumulate reduction, and a mantissa field trades shift-adds for accuracy across a 4-to-8-bit family. Fused kernels deliver up to $4.60\times$ FP16 attention throughput with a $2.5\times$ smaller KV cache, near-FP16 perplexity, and zero-shot accuracy within noise of FP16 at eight stored bits. A cost model calibrated on the disassembled kernels locates the remaining bottleneck in instruction issue rather than memory, and shows that a single architectural addition, a packed 4-way shift-dot-accumulate (DS4A) whose key-side operand is half the width of `__dp4a`'s, would take the same kernel to $6.51\times$. The result is a concrete, quantified request to hardware designers rather than an appeal to the general cheapness of shifters.

Ethics statement

This work concerns the arithmetic and memory behaviour of an inference kernel. It introduces no new dataset and no new model: all experiments use publicly released, openly licensed checkpoints and the public WikiText-103, HellaSwag, PIQA, WinoGrande and ARC corpora, under their respective licences. We note the usual caveat for any efficiency technique: quantisation alters a model's outputs, and Sec. 6.4 shows the change is small on the benchmarks measured but not zero, so a deployment with safety-critical behaviour should re-validate rather than assume the quantised model inherits the original's properties.

REPRODUCIBILITY STATEMENT

The complete implementation is provided to reviewers as anonymised supplementary material: the CUDA kernels, the bit-exact PyTorch reference, the quantiser family, the patched decoder and KV cache, and every evaluation script. The archive also contains the recorded JSON results behind each table, and the scripts that regenerate the tables from them, so each reported number can be traced to the run that produced it. A public repository will be released upon acceptance.

AI USE STATEMENT

Generative AI tools were used solely to correct grammatical errors and improve the clarity and readability of the manuscript. All technical content, analyses, results, interpretations, and conclusions were developed and verified by the authors, who take full responsibility for the final manuscript.

REFERENCES

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 37 (NeurIPS)*, 2024.

Hanting Chen, Yunhe Wang, Chunjing Xu, Boxin Shi, Chao Xu, Qi Tian, and Chang Xu. AdderNet: Do we really need multiplications in deep learning? In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

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

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 35 (NeurIPS)*, 2022.

Mostafa Elhoushi, Zihao Chen, Farhan Shafiq, Ye Henry Tian, and Joey Yiwei Li. DeepShift: Towards multiplication-less neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Cong Guo, Chen Zhang, Jingwen Leng, Zihan Liu, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. ANT: Exploiting adaptive numerical data type for low-bit deep neural network quantization. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022.

Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. OliVe: Accelerating large language models via hardware-friendly outlier-victim pair quantization. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2023.

Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H. Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W. Lee, and Deog-Kyoon Jeong. A^3: Accelerating attention mechanisms in neural networks with approximation. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2020.

Tae Jun Ham, Yejin Lee, Seong Hoon Seo, Soosung Kim, Hyunji Choi, Sung Jun Jung, and Jae W. Lee. ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2021.

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 37 (NeurIPS)*, 2024.

Jungi Lee, Wonbeom Lee, and Jaewoong Sim. Tender: Accelerating large language models via tensor decomposition and runtime requantization. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2024.

Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

Zhikai Li and Qingyi Gu. I-ViT: Integer-only quantization for efficient vision transformer inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17065–17075, 2023.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

Yang Lin, Tianyu Zhang, Peiqin Sun, Zheng Li, and Shuchang Zhou. FQ-ViT: Post-training quantization for fully quantized vision transformer. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1173–1179, 2022.

Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023.

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 (ICML)*, 2024.

Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*, 2017.

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

Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. In *arXiv preprint arXiv:1603.01025*, 2016.

NVIDIA Corporation. *CUDA Math API Reference Manual*. NVIDIA Corporation, release 13.3 edition, June 2026a. URL `https://docs.nvidia.com/cuda/pdf/CUDA_Math_API.pdf`. Chapter 13, Integer Intrinsics.

NVIDIA Corporation. *Parallel Thread Execution ISA*. NVIDIA Corporation, version 9.4 edition, 2026b. URL `https://docs.nvidia.com/cuda/pdf/ptx_isa_9.4.pdf`. Secs. 9.7.1.24 and 9.7.20.1.2: `dp4a` and `vshl`/`vshr`.

Huihong Shi, Xin Cheng, Wendong Mao, and Zhongfeng Wang. P^2-ViT: Power-of-two post-training quantization and acceleration for fully quantized vision transformer. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.

Hanrui Wang, Zhekai Zhang, and Song Han. SpAtten: Efficient sparse attention architecture with cascade token and head pruning. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML)*, 2023.

Haoran You, Xiaohan Chen, Yongan Zhang, Chaojian Li, Sicheng Li, Zihao Liu, Zhangyang Wang, and Yingyan Lin. ShiftAddNet: A hardware-inspired deep network. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

Haoran You, Huihong Shi, Yipin Guo, and Yingyan Lin. ShiftAddViT: Mixture of multiplication primitives towards efficient vision transformer. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023.

Zhihang Yuan, Chenhao Xue, Yiqi Chen, Qiang Wu, and Guangyu Sun. PTQ4ViT: Post-training quantization for vision transformers with twin uniform quantization. In *European Conference on Computer Vision (ECCV)*, 2022.

Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, et al. FlightLLM: Efficient large language model inference with a complete mapping flow on FPGAs. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2024.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate LLM serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

APPENDICES

A QUANTISER DESIGN CHOICES

Table 7 sweeps the remaining axes, and two entries are counter-intuitive enough to be worth stating. A per-key-vector scale is clearly best, and is also the only option needing no calibration pass since it is computed as the key is appended. Clipping estimators that help uniform quantisers (percentile, RMS) degrade here ($0.1302 \rightarrow 0.1717$): the maximum-magnitude element of each key vector maps to $u{=}1$ and is reconstructed exactly, and clipping destroys that exactness for the element contributing most to the dot product. A per-channel exponent bias, which is free in this datapath because it merely offsets the shift index, ($0.1302 \rightarrow 0.2075$), a useful negative result, since per-channel key scaling is standard practice for uniform KV quantisation. Choosing the exponent that minimises absolute rather than log-domain error is a small consistent gain at no hardware cost. The Hadamard rotation is the largest single improvement ($0.1302 \rightarrow 0.0954$), though it helps a uniform INT4 code more (0.0474): rotation Gaussianises the marginals, which suits a uniform grid better than a logarithmic one, so power-of-two codes are at their best on exactly the heavy-tailed distributions rotation removes.

Table 7: Quantiser design choices, three exponent bits throughout, first row of each group the baseline. Estimators and per-channel biases that help uniform codes degrade a logarithmic one.

Axis	Variant	ε_S	top-8
scale granularity	per key vector	0.1302	0.777
	per head	0.2192	0.708
scale estimator	abs. max	0.1302	0.777
	99.9th percentile	0.1302	0.777
	RMS clipping	0.1717	0.746
exponent rounding	nearest in $\log_2$	0.1302	0.777
	error aware	0.1247	0.780
per-channel bias	off	0.1302	0.777
	on	0.2075	0.699
normalisation	none	0.1302	0.777
	Q/K smoothing	0.1273	0.772
	Hadamard rotation	0.0954	0.814

B Implementation notes

Two kernel-level decisions changed the measured speed-up by more than a factor of two, and are worth recording because a shift-based kernel written without them will under-report the arithmetic. First, the code and value tiles must be staged with 16-byte vector accesses; byte- and word-wise staging cost $1.8\times$. Second, the condition $F \geq E$ of (4) should be enforced so the inner loop takes an unconditional left-shift path, worth a further $1.20\times$. Third, the register pressure identified in Sec. 4 can be traded back: constraining the budget with `__launch_bounds__` spills a little but restores enough occupancy to recover a further $1.13\times$. All reported numbers use these settings.

C Both attention products, and why only one is enabled by default

Equations (3) and (9) make both attention MatMuls multiplier-free, and both are implemented and validated in the kernel: with the integer-$\log_2$ softmax (8) enabled, $\mathbf{A}$ is itself a power of two and $\mathbf{AV}$ reduces to shifts. We nonetheless report the shift-based $\mathbf{AV}$ as an option rather than the default, because on a GPU it is a pure loss. It costs $+0.15$ perplexity for PoT-4 ($9.303 \rightarrow 9.450$) and $+0.13$ for PoT-M1 ($8.173 \rightarrow 8.304$), since forcing the attention weights onto a logarithmic grid is a second quantisation on top of the score quantisation; and it returns nothing measurable in speed ($182.6 \rightarrow 183.0$ tokens/s), because the $\mathbf{AV}$ stage is a single streaming pass whose floating-point form is one fused multiply-add per term, again a primitive the hardware already provides at one instruction, exactly the situation of Sec. 4.1. On an FPGA or ASIC the trade-off inverts: the shift-based $\mathbf{AV}$ is what removes the second multiplier array, and the 0.15 perplexity is paid once for a datapath with no multipliers anywhere in attention.