# BEYOND ACCURACY: PREFIX-INVARIANT REALIZATIONS OF LOW-PRECISION FAST MATRIX MULTIPLICATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Fast matrix multiplication saves multiplications through exact cancellation, but rounding sums that mix token rows can leave contributions from later tokens in earlier language model outputs. This threatens prefix invariance, which multiple-choice likelihood scoring relies on: a scored likelihood must depend only on its allowed prefix. On Qwen2.5-14B-Instruct, two fast FP8 realizations repaired to ordinary-looking accuracy still change the answers chosen by likelihood on 5.83% and 10.00% of 240 OpenBookQA items when only the text after the allowed prefix is replaced with the bf16 model's own greedy continuation. Both row-local controls, the bf16 model and a deployed FP8 matrix multiplication kernel, change none. Accuracy thus does not certify prefix invariance, and the stability criteria we analyze cannot tell realizations apart: across all 512 sign variants of two-level Strassen they stay constant while teacher-forced perplexities span a 772.4$\times$ range on the same model. We therefore construct certified realizations of two-level Strassen on bounded integer codes that quantize token rows independently, then mix and cancel exactly before rescaling, using 49 block multiplications instead of 64. Our certificate guarantees bitwise equality to a prescribed row-local classical int8 operator at the same quantization specification, so every certified realization inherits its prefix invariance. Certification thus turns realization choice into a pure cost decision: which certified realization runs can no longer change a single scored likelihood.

## 1 INTRODUCTION

Fast matrix multiplication saves multiplications: it multiplies sums of matrix blocks and lets the unwanted cross terms cancel exactly (Strassen, 1969). Low-precision kernels for large language model (LLM) inference now run such algorithms (Jangda, 2026; Zhu et al., 2026), and in a linear layer the rows of the activation matrix are token positions, so these block sums add rows of different tokens. In exact arithmetic the cross terms cancel and each output row depends only on its own input row; after rounding they can fail to cancel (Malik & Becker, 2019), and a later token can then contribute to an earlier token's output. Such a contribution arises inside a linear layer, where the causal attention mask has no effect. It violates prefix invariance (Kim et al., 2026): what a model computes at a position must depend only on its allowed prefix, the text of its own sequence up to that position. Likelihood-based evaluation relies on this property: it can score all tokens of a candidate answer in one forward pass, and the likelihood of each must depend only on the text before it.

The natural first check on a fast low-precision realization, an algorithm's coefficients together with how it is quantized and executed, is the model's accuracy. In 8-bit floating point (FP8), fast algorithms can lose much of that accuracy (Figure 1A), and two repairs adapted from prior work aim to restore it by changing how the algorithm is quantized (Dumitrescu, 1998; Ballard et al., 2016; Nagel et al., 2020). With both repairs, FP8 realizations of two different fast algorithms bring Qwen2.5-14B-Instruct (Yang et al., 2024) to 35.42% and 32.92% accuracy on OpenBookQA (Mihaylov et al., 2018), against 34.58% for the model in bfloat16 (bf16). Yet when only the text after the allowed prefix is replaced by the bf16 model's own greedy continuation of that prefix, with the original answer still the one scored, they change the answer chosen by likelihood on 5.83% and 10.00% of 240 items, respectively (Figure 1A). Two row-local controls, the bf16 model and a deployed FP8



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.

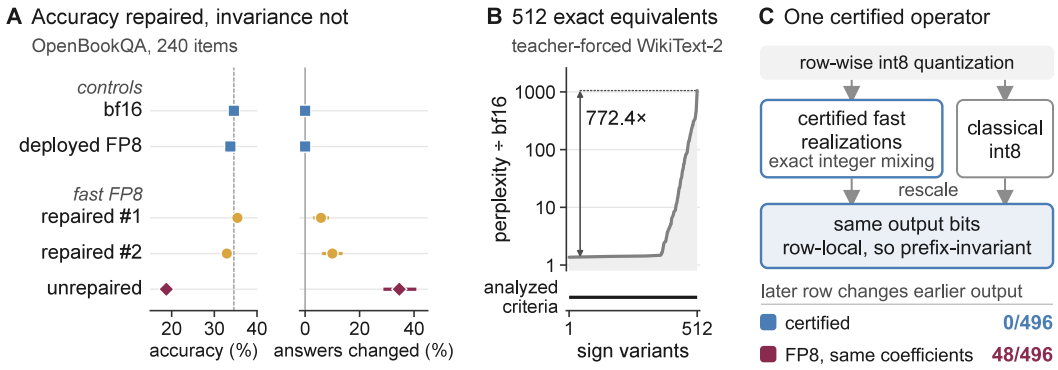


Figure 1: **A** Accuracy on the original input, and answers changed (95% intervals) when the text after the allowed prefix is replaced by the bf16 model's greedy continuation; #1, #2: two fast algorithms, each with both repairs. **B** Perplexity of the 512 sign variants of two-level Strassen under one FP8 schedule, sorted, and the criteria we analyze. **C** A certified realization and classical int8; strip: synthetic row pairs in which a later row changes an earlier output (FP8: our schedule, same coefficients). A, B: Qwen2.5-14B-Instruct.

matrix multiplication kernel, change none. The allowed prefix, input shapes and execution plan are unchanged, so the changed answers expose a dependence on text the answer must not see. The same kind of dependence crosses requests packed into one forward pass, so prefix invariance matters for LLM serving as well as for evaluation.

Since accuracy did not reveal these violations and rounding can produce them, the natural next check is the numerical analysis used to choose among fast algorithms. It compares them by stability criteria computed from their coefficients, such as the parameters of a bound on rounding error (Ballard et al., 2016) or a measure of expected error (Xie et al., 2026). The criteria we analyze read only coefficient magnitudes. Changing an algorithm's coefficient signs in ways that cancel out leaves the exact product unchanged (Malik & Becker, 2019); two-level Strassen, Strassen's algorithm applied to its own blocks, has 512 such sign variants, and each criterion gives all of them one value (Malik & Becker, 2019; Dumas et al., 2026). Under the same FP8 schedule with block scaling, however, their teacher-forced perplexity on the same model spans a 772.4× range (Figure 1B).

Accuracy did not flag the violations above, and the criteria we analyze did not even separate realizations whose quantized behavior differs widely. A test like ours can expose a violation but not prove its absence, so a guarantee that a realization keeps prefix invariance has to come from how it is built. The most direct construction forbids every intermediate product that feeds an output row from depending on a later row, but we prove that this forces the classical number of multiplications (Section 4.1), which gives up the saving. A realization that keeps the saving must therefore contain products that mix later rows into earlier ones, and prefix invariance can then hold only for its final output: the mixing must be undone exactly by the time that output is formed.

We therefore construct realizations of two-level Strassen that undo the mixing exactly, by carrying it out on bounded integer codes, whose sums do not round. Where EmuGEMM uses this exactness to save multiplications through cancellation without rounding error (Ozaki et al., 2026; Lu et al., 2026), we use it to make a fast realization compute exactly what classical int8 matrix multiplication computes. Each token row is quantized to integer codes on its own, the algorithm mixes and cancels these codes, and the scales are applied only afterwards. Our certificate checks that the integer computation stays exact and that the scales can still be applied as in classical int8. Every realization that passes computes, bit for bit, the classical int8 operator with the same codes and scales, rescaled and accumulated in the same order (Figure 1C); since that operator is row-local, every certified realization inherits its prefix invariance. All 512 sign variants of two-level Strassen pass the certificate and compute the same output bits.

The certified realizations match classical int8 bit for bit while using 49 block multiplications instead of 64. Making them run faster than classical int8 is a separate task: Section 5 reports our first

timings and leaves faster kernels to future work. Certification makes the choice between them a decision about cost alone: taking the saving no longer risks prefix invariance.

## 2 REALIZATIONS AND PREFIX INVARIANCE

### 2.1 FAST ALGORITHMS AND THEIR SIGN VARIANTS

A fast matrix multiplication algorithm for $C = AB$ splits $A$ into an $m \times k$ grid of blocks $A_{il}$ and $B$ into a $k \times n$ grid of blocks $B_{lj}$, multiplies $R$ pairs of block sums and combines the results:

$$M_r = \Big(\sum_{i,l} u_{r,il}\, A_{il}\Big)\Big(\sum_{l,j} v_{r,lj}\, B_{lj}\Big) \quad (r = 1, \ldots, R), \qquad C_{ij} = \sum_{r=1}^{R} w_{r,ij}\, M_r. \tag{1}$$

The coefficients $(U, V, W)$ make every unwanted cross term cancel; the classical algorithm uses $R = mkn$ block multiplications, and two-level Strassen, with $m = k = n = 4$, uses 49 instead of 64. Different coefficients can compute the same product. Let $D$, $E$ and $F$ be diagonal sign matrices whose entries $d_i$, $e_l$ and $f_j$ flip whole block rows or columns. Since $AB = D\,(DAE)(EBF)\,F$, running the algorithm on $DAE$ and $EBF$ and flipping the result back computes $AB$ exactly (Malik & Becker, 2019). Absorbing the signs gives the coefficients $d_i e_l u_{r,il}$, $e_l f_j v_{r,lj}$ and $d_i f_j w_{r,ij}$, with the same number of block multiplications; these coefficient sets are the algorithm's sign variants. Negating all of $D$, $E$ or $F$ leaves every weighted product $w_{r,ij} M_r$ unchanged, so we fix $d_1 = e_1 = f_1 = 1$; two-level Strassen then has 512 sign variants.

Quantized and executed the same way, the sign variants are exactly equivalent realizations, yet in low precision they can still differ in accuracy. Choosing among such realizations is an old problem, from the fixed-point digital filters of Mullis & Roberts (1976) to fast matrix multiplication, where the sign variants are a small part of a larger set of equivalences (de Groote, 1978) that Dumas et al. (2026) search for a smaller bound on the rounding error. Malik & Becker (2019) randomize exactly these sign changes and find that random signs can lower the error on synthetic matrices.

### 2.2 QUANTIZATION AND THE REFERENCE OPERATOR

Sign changes can alter the error because they change the values a low-precision realization rounds, and how it rounds them is set by how it is quantized and executed; for integer codes, as in classical int8 and our construction, a quantization specification $c$ spells these rules out. On the way in, $c$ fixes how the operands become integer codes: the code bounds $b_A$ and $b_B$ (the largest code magnitudes); the length $g$ of the groups along the inner dimension that share a scale (block scaling); the rows or columns each scale belongs to; the rule that sets it; and the rounding to codes. On the way out, it fixes how the group results become the output: the precision and exact order of operations in which they are rescaled and summed, and the output format. We treat $c$ as part of the operator a realization computes: changing any of these, even the order of the final additions, can change the output bits.

Our reference, which a certified realization must reproduce, is the classical int8 operator at $c$: it multiplies the codes of each group exactly and rescales and sums the group results as $c$ prescribes. Because $c$ gives each row of $A$ and each column of $B$ its own scale per group, set from that group's entries, row $t$ of the output reads only row $t$ of $A$, its codes and its scales: the operator is row-local.

### 2.3 PREFIX INVARIANCE

Row locality matters for a property that accuracy does not certify: prefix invariance. It requires that for any two inputs that agree on a position's allowed prefix, the text of its own sequence up to that position, the outputs at that position be the same. The causal mask keeps attention from breaking it, but Kim et al. (2026) show that the mask alone does not guarantee it: other operators that combine positions can break it too. A linear layer keeps it when its matrix multiplication is row-local, like the reference operator. The fast FP8 schedules of Section 1 are not: they round the block sums of $A$, and because the rows of $A$ are token positions, a block sum over different $i$ mixes tokens, and in a packed batch it can mix requests.

Prefix invariance differs from deterministic inference, which batch-invariant kernels now provide for LLM serving (He & Thinking Machines Lab, 2025; The SGLang Team, 2025): deterministic

inference asks that a request's output not change with how it is executed, for example with its batch size, while prefix invariance asks which text an output may depend on at a fixed execution plan and input shape. Batch-invariant kernels fix the order of each sum, and order-independent summation (Ahrens et al., 2020) makes the result independent of it; both act on how a fixed set of numbers is added, while a fast algorithm changes what is added, so neither, by itself, keeps later text or another request's text out of an output.

## 3 EXACTLY EQUIVALENT REALIZATIONS ARE NOT INTERCHANGEABLE

Two checks are natural for a fast low-precision realization, the accuracy of the quantized model and the stability criteria computed from an algorithm's coefficients, and neither certifies prefix invariance. Realizations repaired to accuracy close to the bf16 model's still change answers when only text they must not see is replaced (Sections 3.1 and 3.2), and the criteria we analyze do not even separate sign variants whose errors differ widely once quantized (Section 3.3).

### 3.1 ACCURACY REPAIR DOES NOT RESTORE PREFIX INVARIANCE

We run two-level Strassen in every linear layer of Qwen2.5-14B-Instruct (Yang et al., 2024) except the output layer, rounding its block sums to FP8 with the scales that the deployed FP8 matrix multiplication kernel of DeepGEMM uses for its own operands (Zhao et al., 2025). The model then answers only 18.75% of the first 240 OpenBookQA test items (Mihaylov et al., 2018) correctly, against 34.58% in bf16 (Table 1). We build two fast realizations whose accuracy comes back close to the bf16 model's, each by changing both the algorithm and how its block sums are rounded. Repaired #1 runs one level of Strassen, and repaired #2 runs two levels of the seven-multiplication algorithm that the measure of expected error $\Phi$ of Xie et al. (2026) ranks most accurate. Both rescale each token row by a power of two before rows are summed, the outside scaling of Dumitrescu (1998) as Ballard et al. (2016) describe it, and round the weight block sums jointly to reduce the output error, adapting Nagel et al. (2020) (rules in Appendix C.1). The three fast realizations, unrepaired and repaired, are our compositions of DeepGEMM's FP8 routines. We compare them with two row-local controls: the bf16 model and the deployed FP8 kernel, which scales each token row on its own and forms no block sums.

Ordinary-looking accuracy does not show that the repaired realizations keep prefix invariance, so we test the property directly, following the audit design of Kim et al. (2026). Likelihood scoring reads each token of a candidate answer at the position before it, in one forward pass, so an answer's later tokens are in the input when an earlier one is scored. For each answer token we build a second input that keeps the token's allowed prefix and replaces the token itself and all later text. The original token is still scored, from the position before it, in both inputs (Figure 2A), and an answer's score under replacement sums these token scores. The two inputs have the same shape and run under one execution plan (Appendix C), so a computation that keeps prefix invariance scores them identically. The replacement is either filler, unrelated text shared by all answers of an item, or the bf16 model's greedy continuation of the allowed prefix, an ordinary future rather than an adverse one. On all but 7.2% of the answers, the continuation already differs from the real text at the answer's first token. A changed answer is not run-to-run variation: on an identical input the unrepaired realization and repaired #2 repeat their logits exactly, and an independent second run reproduces every chosen answer of repaired #1 and of both controls.

The repaired realizations reach 35.42% and 32.92% accuracy on OpenBookQA, against 34.58% in bf16, yet change the answer chosen by likelihood on 5.83% [2.92, 8.75] and 10.00% [6.25, 13.75] of the items when the text after the allowed prefix is replaced by the greedy continuation (Table 1). The row-local controls change none, under either replacement. Under the same continuation, target-token log probabilities change at 80.14% and 89.00% of scored positions for repaired #1 and #2, respectively (Table 4). The deployed FP8 kernel rounds to the same format, so the contrast lies within one FP8 format: unlike the controls, the repaired realizations round sums that mix token rows. Table 1 shows the changes persisting under the filler, both on OpenBookQA and on ARC-Easy (Clark et al., 2018).

For repaired #1 on OpenBookQA, the changes under the greedy continuation barely move accuracy: of its 14 changed answers, 9 are wrong before and after the change, 3 go from correct to wrong

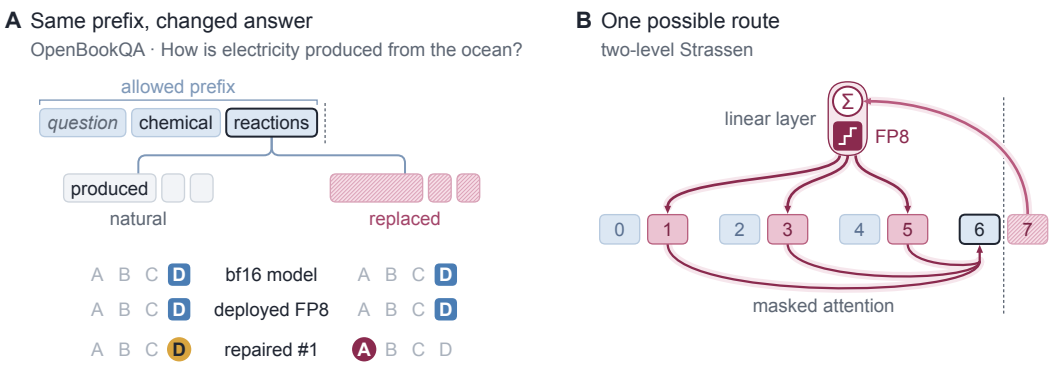


Figure 2: **A** One OpenBookQA item on Qwen2.5-14B-Instruct: the answer chosen with the natural text and with the bf16 model's greedy continuation after an illustrative cut; "produced" is scored at the outlined token. **B** In two-level Strassen, block sums rounded to FP8 carry row 7 into rows 1, 3 and 5 (repaired #1 only into 3), and masked attention carries those rows on to position 6.

Table 1: Accuracy (%) on the original input, and answers changed (%) when the text after the allowed prefix is replaced by the bf16 model's greedy continuation or by filler, with 95% bootstrap intervals over items for the answers changed; Qwen2.5-14B-Instruct, the first 240 test items of each task.

| | OpenBookQA | | | ARC-Easy | |
|---|---|---|---|---|---|
| | | answers changed | | | changed |
| | accuracy | continuation | filler | accuracy | filler |
| bf16 model | 34.58 | 0.00 | 0.00 | 84.17 | 0.00 |
| deployed FP8 kernel | 33.75 | 0.00 | 0.00 | 83.33 | 0.00 |
| repaired #1 | 35.42 | 5.83 [2.92, 8.75] | 7.08 [4.17, 10.42] | 82.50 | 6.67 [3.75, 10.00] |
| repaired #2 | 32.92 | 10.00 [6.25, 13.75] | 7.92 [4.58, 11.67] | 80.83 | 12.08 [8.33, 16.25] |
| unrepaired | 18.75 | 34.58 [28.75, 40.83] | 37.92 [31.67, 44.17] | 39.58 | 60.00 [53.75, 66.25] |

and 2 from wrong to correct. Figure 2A shows one of them, a switch from one wrong answer to another, while both controls keep theirs. Accuracy is therefore blind to most of these changes, and so is a count of changes in correctness, such as the flips that Dutta et al. (2024) use to compare a compressed model with its original.

### 3.2 How the violation can arise and how far it extends

Later text can reach a scored position through the linear layers, where the causal mask has no effect. Two-level Strassen splits the rows of a linear layer's input, which are token positions, into four contiguous blocks after padding with zeros, and its block sums add rows at the same offset in different blocks (Figure 2B). In exact arithmetic the other rows cancel from every output row; after FP8 rounding they need not (Malik & Becker, 2019), so an output row can depend on the later rows at its offset, as a test of a single multiplication confirms (Appendix C.3). Repaired #2 adds the same rows and repaired #1, with one level, adds pairs of them; rescaling and joint rounding change the scale and rounding of these sums, not which rows they add. Later text can thus reach a scored token directly, when a later row shares its offset, or indirectly: it can change earlier rows, and correctly masked attention can carry those rows forward to the scored token.

The dependence is not confined to multiple-choice scoring or to one model: on five models of four families (Yang et al., 2024; 2025; Grattafiori et al., 2024; OLMo et al., 2024), replacing all but the first quarter of each 1,024-token window changes 18.2–40.9% of the next-token predictions within that quarter under the unrepaired realization, 7.8–15.2% under the repaired ones, and none under the controls (Appendix C).

The same dependence crosses requests: we pack two requests into one forward pass of fixed shape, with attention separated between them and no key–value cache, and replace the second request's text with other text of the same length. Over 768 such pairs on three models, the first request's next-token distribution changes in every pair under the unrepaired realization and under repaired #2, and never under the controls. Under repaired #2 its greedy next token changes in 11.9–13.0% of the pairs, and on Qwen2.5-3B the first request's key and value projections already differ at the first layer, before any attention, but not under the controls (Appendix C).

### 3.3 What the stability criteria cannot see

The second check comes from the numerical analysis of fast algorithms, which studies their rounding error, where violations of prefix invariance can arise, and compares algorithms by stability criteria computed from their coefficients. The coefficients alone, however, do not decide prefix invariance: with the same coefficients, two-level Strassen keeps it in exact arithmetic and on bounded integer codes, whose sums do not round (Section 4), and breaks it under the FP8 schedule of Section 3.1.

The criteria we analyze are $\Phi$, the number of nonzero coefficients, the largest coefficient magnitude, and the parameters $Q$ and $E$ of the bound on rounding error of Ballard et al. (2016) (defined in Appendix B.1), and they do not even separate the sign variants of one algorithm by their error. They read the coefficients only through their absolute values or their nonzero pattern, which a sign change leaves unchanged, so all 512 sign variants of two-level Strassen get one value; Malik & Becker (2019, Proposition 10) and Dumas et al. (2026, Lemma 26) prove this for the criteria in their own error bounds: the bounds stay valid, but equal bounds do not promise equal realized error.

Measuring the realized error on random matrices, as the checks that accompany fast kernels and their error analyses do (Jangda, 2026; Dai & Lim, 2023; Lu et al., 2026), would not separate the sign variants either when the entries are independent. These checks draw entries symmetric about zero, and for independent entries a sign variant computes, up to exact sign flips of the output, what the original algorithm computes on $DAE$ and $EBF$, which are distributed exactly as $A$ and $B$ (Section 2.1), so all variants' error norms have the same distribution. With entries uniform on $[0, 1]$ the symmetry breaks and the variants' errors differ (Appendix B.1): how they compare depends on the operands, so we run every variant inside a model, on its own activations.

Under the FP8 schedule of Section 3.1, in every linear layer of Qwen2.5-14B-Instruct except the output layer, we score each of the 512 sign variants by its teacher-forced perplexity on WikiText-2 (Merity et al., 2017), divided by the perplexity of the model in bf16. This ratio runs from 1.367 to 1056.0, a 772.4× spread (Figure 1B), and the variant with Strassen's original signs, the unrepaired realization of Section 3.1, sits at 63.11×. A single sign, which none of the criteria reads, can make nearly all of this difference: two variants that differ only in one of the signs $d_i$, $e_l$, $f_j$ of Section 2.1 have ratios 761.0 times apart. Appendix B reports the spread on seven models, with three controls.

Teacher forcing reads every position in one forward pass, with the text after a position, including the token it predicts, in the input; under a realization that is not row-local, a position's score can therefore depend on that text. Replacing that text with filler chosen independently of the predicted token, for four variants across the range (the smallest, the median, Strassen's original signs and the largest), leaves their spread at 0.985 times its teacher-forced value on the same scored positions (95% confidence interval [0.863, 1.141], bootstrap over text windows; Appendix B.3): the spread survives the replacement.

Exactly equivalent realizations can thus differ once quantized, in what their outputs depend on and in how large their error is; accuracy misses the first and the criteria we analyze miss both.

## 4 Certified realizations on bounded integer codes

Prefix invariance constrains a realization's outputs, not each product it forms on the way.

### 4.1 Prefix-safe products force the classical count

Whether a product keeps later rows out of the outputs it feeds can be read off the coefficients of (1), because the rows of $A$ are token positions, split into $m$ contiguous blocks (Section 3.2). A product

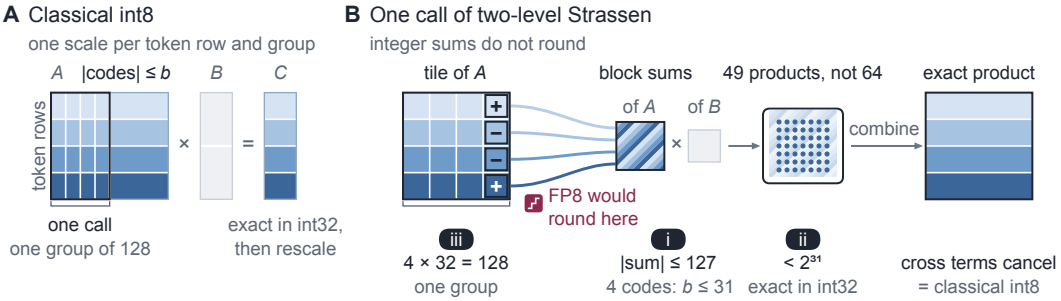


Figure 3: **A** Classical int8 on one group of 128. **B** One call of two-level Strassen on the same codes; (i)–(iii) mark where the conditions of Proposition 1 bind.

$M_r$ reads block row $i$ of $A$ if $u_{r,il} \neq 0$ for some $l$, and feeds block row $i$ of $C$ if $w_{r,ij} \neq 0$ for some $j$. We call an algorithm prefix-safe product by product if no product reads a block row that comes after a block row it feeds. Such an algorithm keeps later rows out of earlier outputs even when each row of its block sums is rounded with its own scale, but it cannot keep the saving.

**Theorem 1** *An algorithm of the form (1) that is prefix-safe product by product uses $R \geq mkn$ block multiplications, and the classical algorithm attains the bound (proof in Appendix A).*

Two-level Strassen, with 49 products instead of 64, therefore has products that carry a later block row into an earlier output, and the contributions of those rows must vanish from every earlier output by the time it is formed, as they do in exact arithmetic.

### 4.2 Computing classical int8 with a fast algorithm

Our construction builds on classical int8 matrix multiplication, which already multiplies integer codes without rounding. At a quantization specification $c$ (Section 2.2), it gives each row of $A$ and each column of $B$ its own scale for every group of $g$ consecutive indices of the inner dimension. It multiplies the codes of each group exactly, and only then applies the scales and adds the group results in the order $c$ fixes (Figure 3). Our realizations keep all of this except the exact product of the codes, which they compute with the fast algorithm on one submatrix, or tile, of $A$ and one of $B$ at a time; we call each such computation a call. Their output bits are those of classical int8 if every call returns its integer product exactly and if the scales can still be applied after it. Rows can keep their own scales, since an exact product builds each output row from its own row's codes alone. The inner dimension is another matter: every fast algorithm mixes its indices too, since one whose products each read a single block column of $A$ needs $mkn$ of them (Theorem 2), and independent scales along it cannot in general be applied after products that do not depend on them (Appendix A).

Our certificate consists of three conditions that secure both requirements, and we call a realization that meets them certified. They involve the largest coefficient 1-norms of the block sums, $L_A = \max_r \sum_{i,l} |u_{r,il}|$ and $L_B = \max_r \sum_{l,j} |v_{r,lj}|$, the largest total weight $L_W = \max_{i,j} \sum_r |w_{r,ij}|$ with which an output block combines products, and the inner length $h$ of a block.

**Proposition 1** *Let a realization quantize, rescale and accumulate as the classical int8 operator at a quantization specification $c$ does, and compute the integer product of the codes tile by tile, each call with an algorithm of the form (1) with integer coefficients. Suppose that (i) $L_A b_A \leq 127$ and $L_B b_B \leq 127$; (ii) $\max\{L_W h(L_A b_A)(L_B b_B),\ g b_A b_B\} < 2^{31}$ for int32 accumulation, or $< 2^{24}$ for exact integers in fp32; and (iii) each call lies within one group along the inner dimension. Then on every input the realization computes the output of the classical int8 operator at $c$, bit for bit.*

Conditions (i) and (ii) secure the first requirement: every block sum of codes stays within int8, so the products run on the same int8 multipliers as classical int8, and every product and partial sum, the running sum over a group's calls included, stays within the accumulator. Condition (iii) secures the second, since a call that stays within one group contributes only to that group's integer product, which is then rescaled as in classical int8. The conditions involve only the coefficients, the tiling

and $c$, so they are checked once, before any input is seen. Every certified realization is therefore row-local, like classical int8, and keeps prefix invariance on every input (Section 2.3). Two-level Strassen has $L_A = 4$ and $L_B = 4$, so condition (i) admits codes up to $b_A = 31$ and $b_B = 31$. With $L_W = 16$, int32 accumulation is exact for blocks of inner length up to 8729, and with our groups of $g = 128$ we use blocks of inner length 32, so the four block columns of a call span one group. Two-level Strassen is thus certified at code bound 31, with 49 block multiplications, 0.7656 of the classical count.

Our integer product equals the classical one bit for bit on 200 random tiles of codes; on a separately drawn tile the 512 sign variants give 1 distinct integer product, where rounding the block sums to FP8 leaves 0 of them exact and a 3.17-fold spread in the largest error, and with the scales applied as well, under the full quantization specification on inputs of 32 token rows in four blocks, a subset of 64 variants again gives 1 distinct output.

We then test prefix invariance directly, on separately drawn Gaussian inputs of that shape: changing one token row at a time and comparing every earlier output row moves 0 of the 496 pairs of an earlier and a later row. The same coefficients move 48 pairs under an FP8 schedule that rounds the block sums before multiplying, exactly those whose rows share an offset in their blocks (Figure 2B). A control that breaks condition (iii), with scales on groups of 32 indices while a call spans 128, rounds each mixed block sum again with one scale, set by its largest magnitude, which any of its rows can change, and all 496 pairs move.

Inside a model, we run two-level Strassen at code bound 31 in every linear layer except the output layer of four models, Qwen2.5-14B-Instruct, Llama-3.1-8B-Instruct, OLMo-2-7B-Instruct and Qwen3-8B, emulating its int8 arithmetic exactly in fp32, where every intermediate stays below $2^{24}$. Each time a replaced layer runs, its output matches classical int8's bit for bit (Appendix D), so the remaining model measurements run that classical int8 operator, which gives the same outputs. On the test that exposed the repaired FP8 realizations, replacing the text after the allowed prefix with the greedy continuation (Section 3.1), the operator changes none of the 240 OpenBookQA answers on any of the four models, like the row-local controls.

### 4.3 Full-range codes with an overflow correction

Certification on int8 multipliers has a price, and condition (i) sets it: a realization certified by condition (i) can use codes only up to $\lfloor 127/L_A \rfloor$ in $A$ and $\lfloor 127/L_B \rfloor$ in $B$, 31 for two-level Strassen (Appendix D).

The overflow behind this price can be corrected instead of avoided. Each block sum of codes $\hat{A}$ splits exactly into $\hat{A} = A_0 + 256R_A$, with $A_0$ the remainder in the int8 range and $R_A$ nonzero only where $\hat{A}$ overflows, and likewise $\hat{B} = B_0 + 256R_B$, so that

$$\hat{A}\hat{B} = A_0B_0 + 256\,(R_AB_0 + A_0R_B) + 256^2R_AR_B. \tag{2}$$

Two-level Strassen then forms its products on $A_0$ and $B_0$ and adds the correction terms of (2), which take further products, counted in Section 5. With this correction it computes classical int8 at code bound 127 bit for bit at the same groups of 128, under a certificate of its own (Appendix D).

### 4.4 Model quality and the choice of realization

Bitwise identity fixes which operator a certified realization computes, not how close that operator comes to the bf16 model. We measure that closeness for the operators the realizations compute, classical int8 at code bound 31 and, with the correction, at 127, by their excess negative log-likelihood (NLL) per token over the bf16 model on eight chunks of WikiText-2, against the deployed FP8 kernel (Figure 4). Since a change of temperature alone can move such a comparison (Guo et al., 2017), we report the excess both raw, at temperature 1, and adjusted, with every operator and the bf16 model at a temperature fitted without the scored chunk. The raw excess, which scores the models as they are run, is the primary reading. At code bound 31 the excess is larger than the kernel's on three of the four models raw, all but Qwen3-8B, and on all four adjusted. With the overflow correction, at code bound 127, the excess is at or below the kernel's on all four models, raw and adjusted. Table 8 also reports ordinary OpenBookQA accuracy for these operators.

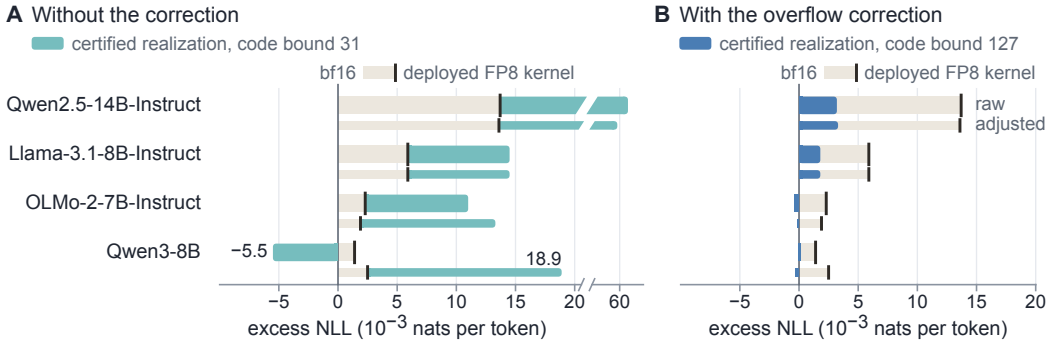


Figure 4: Excess NLL per token over bf16 on eight 2048-token chunks of WikiText-2, at code bound 31 (**A**) and at 127 with the overflow correction (**B**); thick bars raw, thin bars with temperatures fitted on the other seven chunks, bands ending at the deployed FP8 kernel's value. Certified values are measured on the classical int8 operators the realizations compute (Table 7).

## 5 Cost and future work

A certified realization and classical int8 differ in two costs, the multiplications they perform and the time they take. In multiplications, two-level Strassen at code bound 31 performs 0.7656 of classical int8's multiplications. Raising the code bound to 127 brings the excess NLL to or below the deployed FP8 kernel's on all four models (Section 4.4), but the correction of Section 4.3 then adds products wherever a block sum overflows. If the correction multiplies only the entries that overflow, the corrected computation performs 0.79–0.82 of classical int8's multiplications on sampled layer slices from three models (Appendix E); if it multiplies every row of $R_A$ and column of $R_B$ that holds an overflow, it performs 1.43–1.71 times as many. Whether the full code range keeps the saving thus depends on how the correction is implemented.

In time, the kernels we have built for the certified realizations without the correction are slower than classical int8 at the shape we timed, two $4096 \times 4096$ matrices on an NVIDIA H20 graphics processing unit (GPU). With our Triton kernels (Tillet et al., 2019), one level of Strassen takes 2.04–3.17 times as long as classical int8 across three group sizes, and two levels take 27.9 times as long; part of this time goes to splitting a call into short block products, which slows even the classical algorithm (Appendix E). It remains open whether, and how far, the saving in multiplications can be turned into speed. Any optimization that preserves the quantization specification and remains certified yields the same output bits. We leave faster kernels, including one for the correction, and their timing across shapes and GPUs to future work.

## 6 Conclusion

A fast matrix multiplication inside a language model has to do more than stay accurate: it has to keep prefix invariance, which likelihood-based evaluation and LLM serving rely on. The two can come apart. A low-precision fast realization repaired to accuracy close to the bf16 model's can still let its outputs depend on text they must not see, so prefix invariance has to be certified directly. Such a certificate need not cost the saving in multiplications. Although a fast algorithm cannot save multiplications without products that mix the rows of different tokens, prefix invariance constrains only the output those products produce. A realization whose output is exactly that of a row-local operator keeps prefix invariance, whatever its products mix. Two-level Strassen, which uses 49 block multiplications instead of 64, admits realizations of this kind: the ones we certify on bounded integer codes compute the row-local classical int8 operator bit for bit at the same quantization specification. Without certification, exactly equivalent realizations can compute different operators once quantized, so choosing one can change what the model computes. With it, which operator to run is a question of the quality it gives the model, and which certified realization computes it is a pure cost decision.

### AI USE STATEMENT

In this work, we used generative AI tools to assist with research ideation and execution, including implementing the methods, running the experiments and analyzing their results, and to assist in proving the mathematical claims. We have not used generative AI tools to generate synthetic data sets or for translation, and cleaning or reformatting data sets and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to find related work and to help draft and polish the text. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### REPRODUCIBILITY STATEMENT

Section 2.2 defines the quantization specification and the classical int8 operator that a certified realization reproduces. Appendix A gives the proofs for Section 4 with their assumptions, and Appendix D gives the certificate of the computation with the overflow correction. Appendices B to E describe the experimental protocols, the controls and the cost measurements. Appendix F maps every family of results to the arithmetic and hardware that produced it and describes how the printed numbers are generated from stored program outputs. The anonymous supplementary material contains the code of the methods and experiments and their stored outputs, with instructions for rerunning the experiments.

## REFERENCES

Willow Ahrens, James Demmel, and Hong Diep Nguyen. Algorithms for efficient reproducible floating point summation. *ACM Transactions on Mathematical Software*, 46(3):1–49, 2020. doi: 10.1145/3389360.

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 Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10–15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/b5b939436789f76f08b9d0da5e81af7c-Abstract-Conference.html.

Grey Ballard, Austin R. Benson, Alex Druinsky, Benjamin Lipshitz, and Oded Schwartz. Improving the numerical stability of fast matrix multiplication. *SIAM Journal on Matrix Analysis and Applications*, 37(4):1382–1418, 2016. doi: 10.1137/15M1032168.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018.

Zhen Dai and Lek-Heng Lim. Numerical stability and tensor nuclear norm. *Numerische Mathematik*, 155:345–376, 2023. doi: 10.1007/s00211-023-01377-5.

Hans F. de Groote. On varieties of optimal algorithms for the computation of bilinear mappings. II. optimal algorithms for $2 \times 2$-matrix multiplication. *Theoretical Computer Science*, 7(2):127–148, 1978. doi: 10.1016/0304-3975(78)90045-2.

Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. Dense linear algebra over word-size prime fields: the FFLAS and FFPACK packages, 2009. Version v3, 14 January 2009. Journal version: ACM Trans. Math. Softw. 35(3), 2008, pp. 1–42, doi:10.1145/1391989.1391992.

Jean-Guillaume Dumas, Clément Pernet, and Alexandre Sedoglavic. Towards automated generation of fast and accurate algorithms for recursive matrix multiplication. *J. Symb. Comput.*, 134: 102524, 2026. doi: 10.1016/J.JSC.2025.102524. URL https://doi.org/10.1016/j.jsc.2025.102524. Preprint: arXiv:2506.19405.

Bogdan Dumitrescu. Improving and estimating the accuracy of strassen's algorithm. *Numerische Mathematik*, 79(4):485–499, June 1998. ISSN 0945-3245. doi: 10.1007/s002110050348. URL http://dx.doi.org/10.1007/s002110050348.

Abhinav Dutta, Sanjeev Krishnan, Nipun Kwatra, and Ramachandran Ramjee. Accuracy is not all you need, 2024. URL https://arxiv.org/abs/2407.09141.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, October 2022. ISSN 1476-4687. doi: 10.1038/s41586-022-05172-4. URL http://dx.doi.org/10.1038/s41586-022-05172-4.

Aaron Grattafiori et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks, 2017. URL https://arxiv.org/abs/1706.04599.

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

Abhinav Jangda. Compiling strassen-like matrix multiplication algorithms to fast cuda kernels. *Proceedings of the ACM on Programming Languages*, 10(PLDI):505–530, June 2026. ISSN 2475-1421. doi: 10.1145/3808267. URL http://dx.doi.org/10.1145/3808267.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Taebong Kim, Youngsik Hong, Minsik Kim, Sunyoung Choi, Jaewon Jang, and Minseo Kim. The mask is not the model: Auditing prefix invariance in attention, state-space, and hybrid sequence models, 2026. URL https://arxiv.org/abs/2608.22876.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: LLM quantization with learned rotations. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=ogO6DGE6FZ.

Denghui Lu, Alexander Maeder, Mathieu Luisier, and Alexandros Nikolaos Ziogas. Emugemm: Fused tensor core kernels for precision emulation in matrix multiplication, 2026. URL https://arxiv.org/abs/2606.25453.

Osman Asif Malik and Stephen Becker. Randomization of approximate bilinear computation for matrix multiplication, 2019. URL https://arxiv.org/abs/1905.07439.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018. URL https://arxiv.org/abs/1809.02789.

Pierpaolo Mori, Lukas Frickenstein, Shambhavi Balamuthu Sampath, Moritz Thoma, Nael Fasfous, Manoj Rohit Vemparala, Alexander Frickenstein, Christian Unger, Walter Stechele, Daniel Mueller-Gritschneder, and Claudio Passerone. Wino vidi vici: Conquering numerical instability of 8-bit winograd convolution for accurate inference acceleration on edge. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 53–62, January 2024. URL https://openaccess.thecvf.com/content/WACV2024/html/Mori_Wino_Vidi_Vici_Conquering_Numerical_Instability_of_8-Bit_Winograd_Convolution_WACV_2024_paper.html.

C. Mullis and R. Roberts. Synthesis of minimum roundoff noise fixed point digital filters. *IEEE Transactions on Circuits and Systems*, 23(9):551–562, September 1976. ISSN 0098-4094. doi: 10.1109/tcs.1976.1084254. URL http://dx.doi.org/10.1109/TCS.1976.1084254.

Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization, 2020. URL https://arxiv.org/abs/2004.10568.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious. *arXiv.org*, 2024. doi: 10.48550/arXiv.2501.00656.

Katsuhisa Ozaki, Yuki Uchino, and Toshiyuki Imamura. Ozaki scheme ii: A gemm-oriented emulation of floating-point matrix multiplication using an integer modular technique. *The International Journal of High Performance Computing Applications*, 2026. doi: 10.1177/10943420261467787.

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, Zach 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, 2019. URL https://arxiv.org/abs/1912.01703.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019. URL https://arxiv.org/abs/1910.10683.

Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. doi: 10.1007/BF02165411.

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

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*, pp. 10–19. ACM, 2019. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi,

Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. *arXiv.org*, 2019.

Shuxiao Xie, Shuyang Xie, Yuan Cao, Dezhi Ran, Wei Yang, and Tao Xie. Fast matrix multiplication in fp8: Certified coefficient optimization and measured error, 2026. URL https://arxiv.org/abs/2609.26077.

An Yang et al. Qwen2.5 technical report, 2024.

An Yang et al. Qwen3 technical report, 2025.

Ziteng Yang, Nicholas J. Riasanovsky, Warren Deng, and Vivek Sarkar. Taming bitwise behavior in gpu kernels with tensor core: Black-box reconstruction, compiler enforcement, and static verification, 2026. URL https://arxiv.org/abs/2609.11356.

Alex Young et al. Yi: Open foundation models by 01.ai, 2024.

Chenggang Zhao, Zhean Xu, Liang Zhao, Jiashi Li, Chenhao Xu, Anyi Xu, Shengyu Liu, Kexing Zhou, and Kuai Yu. DeepGEMM: clean and efficient BLAS kernel library on GPU. https://github.com/deepseek-ai/DeepGEMM, 2025.

Honglin Zhu, Jiaping Cao, Jiang Shao, Siyuan Feng, Qian Qiu, Peng Chen, Xu Zhang, Yixian Zhou, Man Lung Yiu, Guang Ji, Minwen Deng, Jintao Meng, and Wenxi Zhu. Falcongemm: Surpassing hardware peaks with lower-complexity matrix multiplication, 2026. URL https://arxiv.org/abs/2605.06057.

# A Proofs for Section 4

## A.1 Prefix-safe products force the classical count

Within one sequence, the most direct way to guarantee prefix invariance is to require every product to keep later rows out of the outputs it feeds, and Theorem 1 states what that costs: an algorithm of the form (1) that is prefix-safe product by product, so that no product reads a block row of $A$ that comes after a block row of $C$ it feeds, uses $R \geq mkn$ block multiplications, as many as the classical algorithm. The proof counts, for each block row, the products that both read it and feed it, and shows that prefix safety forbids two block rows to share such a product.

The count depends only on the coefficients, which also give an algorithm for single-entry blocks. Setting every block $A_{il} = x_{il}X$ and $B_{lj} = y_{lj}Y$, for fixed matrices $X$ and $Y$ with $XY \neq 0$, makes every term of (1) a multiple of $XY$, and comparing the multiples shows that the coefficients multiply the $m \times k$ matrix $(x_{il})$ by the $k \times n$ matrix $(y_{lj})$ with $R$ multiplications of numbers. We may therefore take $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{k\times n}$, so that block rows and block columns become rows and columns, and product $r$ is the number $M_r = f_r(A)\, g_r(B)$ with $f_r(A) = \sum_{i,l} u_{r,il} A_{il}$ and $g_r(B) = \sum_{l,j} v_{r,lj} B_{lj}$. We write $E_{il}$ for the matrix, of the size the context requires, whose only nonzero entry is a 1 in position $(i, l)$, and $I$ for the identity matrix.

*Proof of Theorem 1.* For each row $i$, let $\mathcal{R}_i$ be the set of products that read row $i$ and feed row $i$. Every algorithm of the form (1), prefix-safe or not, has at least $kn$ products in each $\mathcal{R}_i$. To see this, take $A = E_{il}$: then $f_r(A) = u_{r,il}$, and row $i$ of $AB$ is row $l$ of $B$, so that

$$B_{lj} = (AB)_{ij} = \sum_{r=1}^{R} u_{r,il}\, w_{r,ij}\, g_r(B) \qquad \text{for every } B.$$

A term of this sum can be nonzero only if product $r$ reads row $i$ and feeds it, that is, only if $r \in \mathcal{R}_i$. Letting $l$ and $j$ vary, each of the $kn$ coordinate functions $B \mapsto B_{lj}$ is therefore a linear combination of the functions $g_r$ with $r \in \mathcal{R}_i$; since the coordinate functions are linearly independent, $\mathcal{R}_i$ has at least $kn$ members. Prefix safety makes the sets disjoint: a product in both $\mathcal{R}_i$ and $\mathcal{R}_{i'}$, with $i < i'$, feeds row $i$ and reads the later row $i'$. Hence $R \geq |\mathcal{R}_1| + \cdots + |\mathcal{R}_m| \geq mkn$. The classical algorithm attains the bound, since its product $A_{il}B_{lj}$ reads row $i$ and feeds only row $i$. □

Strassen's algorithm, with $m = k = n = 2$, shows the sharing that the bound rules out. Its seven products, with $C_{11} = M_1 + M_4 - M_5 + M_7$, $C_{12} = M_3 + M_5$, $C_{21} = M_2 + M_4$ and $C_{22} = M_1 - M_2 + M_3 + M_6$, read and feed these rows:

| | product | reads rows of $A$ | feeds rows of $C$ |
|---|---|---|---|
| $M_1$ | $(A_{11} + A_{22})(B_{11} + B_{22})$ | 1, 2 | 1, 2 |
| $M_2$ | $(A_{21} + A_{22})\, B_{11}$ | 2 | 2 |
| $M_3$ | $A_{11}\, (B_{12} - B_{22})$ | 1 | 1, 2 |
| $M_4$ | $A_{22}\, (B_{21} - B_{11})$ | 2 | 1, 2 |
| $M_5$ | $(A_{11} + A_{12})\, B_{22}$ | 1 | 1 |
| $M_6$ | $(A_{21} - A_{11})(B_{11} + B_{12})$ | 1, 2 | 2 |
| $M_7$ | $(A_{12} - A_{22})(B_{21} + B_{22})$ | 1, 2 | 1 |

Four products read and feed row 1, $M_1$, $M_3$, $M_5$ and $M_7$, and four read and feed row 2, $M_1$, $M_2$, $M_4$ and $M_6$, as the proof requires. The two sets share one product, $M_1$, which is how seven products meet a count of eight. A product shared by two rows always breaks prefix safety, since it feeds the earlier row and reads the later one: $M_1$ feeds row 1 and reads row 2.

### A.2 The inner dimension and its scales

Theorem 1 forces a fast algorithm's products to mix rows. They must also mix indices of the inner dimension, where classical int8 applies the scales of its groups. A product reads block column $l$ of $A$ if $u_{r,il} \neq 0$ for some $i$, and block row $l$ of $B$ if $v_{r,lj} \neq 0$ for some $j$.

**Theorem 2** *An algorithm of the form (1) in which no product reads two block columns of $A$ uses $R \geq mkn$ block multiplications, and so does one in which no product reads two block rows of $B$.*

This time the count runs over the output: on inputs that use only one index of the inner dimension, only the products that read it contribute, and they must produce each of the $mn$ entries of $AB$.

*Proof.* With the single-entry blocks of Appendix A.1, write $W_r$ for the $m \times n$ matrix of the weights $w_{r,ij}$, so that (1) reads $AB = \sum_r f_r(A)\, g_r(B)\, W_r$. Let $\mathcal{Q}_l$ be the set of products that read column $l$ of $A$; under the first hypothesis the sets $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ are disjoint. For $A = E_{il}$ and $B = E_{lj}$, $AB = E_{ij}$, while $f_r(A)\, g_r(B) = u_{r,il}\, v_{r,lj}$ is zero unless $r \in \mathcal{Q}_l$. Each of the $mn$ matrices $E_{ij}$ is therefore a linear combination of the $W_r$ with $r \in \mathcal{Q}_l$, and since the $E_{ij}$ are linearly independent, $\mathcal{Q}_l$ has at least $mn$ members. Summing over the $k$ disjoint sets gives $R \geq mkn$. The sets of products that read row $l$ of $B$ give the second statement in the same way. □

An algorithm with fewer than $mkn$ block multiplications therefore has a product that reads two block columns of $A$, and one that reads two block rows of $B$: some of its block sums add entries from different indices of the inner dimension. Under $c$, those indices carry scales: an output entry's scale is its row's scale times its column's scale, shared by the $g$ indices of each group, and classical int8 applies it after multiplying the group's codes. Suppose that each block column of a call lay in a group of its own, as in the control of Section 4.2, whose groups of 32 indices give each block column of a call spanning 128 indices its own scale. The call forms its products from the codes, the scales of its block columns vary independently of one another, and to reproduce classical int8 it would have to apply these scales after its products. In the lemma below the products are fixed, while the coefficients that combine them may depend on the scales in any way; applied with single-entry blocks, it shows that for arbitrary codes and scales this takes the classical count.

**Lemma 1** *Let $M_1, \ldots, M_R$ be bilinear forms in $(A, B) \in \mathbb{R}^{m\times k} \times \mathbb{R}^{k\times n}$, such as the products of an algorithm of the form (1). Suppose that for every diagonal matrix $\Lambda$ with positive diagonal*

*entries there are real numbers* $\gamma_{ij,r}(\Lambda)$ *such that* $(A\Lambda B)_{ij} = \sum_{r=1}^{R} \gamma_{ij,r}(\Lambda)\, M_r(A,B)$ *for all* $A$*,* $B$*,* $i$ *and* $j$*. Then* $R \geq mkn$*.*

*Proof.* Both $\Lambda = I$ and $\Lambda = I + E_{ll}$ have positive diagonals, and $(A(I + E_{ll})B)_{ij} - (AB)_{ij} = A_{il}B_{lj}$, so each of the $mkn$ forms $A_{il}B_{lj}$ is a linear combination of $M_1, \ldots, M_R$. These forms are distinct monomials and hence linearly independent, so $R \geq mkn$. □

Condition (iii) of Proposition 1 keeps each call within one group, where each output entry's scale is constant along the call's inner dimension: the call's exact integer product can then join the group's integer product before the group is rescaled, as in classical int8.

### A.3 Proof of Proposition 1

Theorems 1 and 2 leave a fast realization no choice but to mix rows and, within a call, indices of the inner dimension. Proposition 1 shows that on bounded integer codes it can still compute the output bits of classical int8, under three conditions: (i) $L_A b_A \leq 127$ and $L_B b_B \leq 127$; (ii) $\max\{L_W h(L_A b_A)(L_B b_B),\, g b_A b_B\} < 2^{31}$ for int32 accumulation, or $< 2^{24}$ for exact integers in fp32; and (iii) each call lies within one group along the inner dimension.

*Proof.* The realization quantizes, rescales and accumulates as the classical int8 operator at $c$ does: it converts each group's integer product into the precision that $c$ fixes, as classical int8 converts the same integer, and rescales and sums the results in the order that $c$ fixes. Computing the integer product tile by tile means that the tiles partition the inner dimension, and by condition (iii) each call lies within one group, so the calls of a group add up to that group's integer product, the exact product of the codes over its $g$ indices of the inner dimension. The realization therefore differs from classical int8 only in how it computes these integers, and it suffices to show that they come out exact.

Consider first one call, on tiles of codes of magnitude at most $b_A$ in $A$ and $b_B$ in $B$, split into blocks of inner length $h$, and write $\alpha = L_A b_A$ and $\beta = L_B b_B$. Entry by entry, for every product $r$, every output block $(i, j)$ and every set $S$ of products,

$$\left|\sum_{i,l} u_{r,il} A_{il}\right| \leq \alpha \leq 127, \qquad \left|\sum_{l,j} v_{r,lj} B_{lj}\right| \leq \beta \leq 127,$$

$$|M_r| \leq h\,\alpha\beta, \qquad \left|\sum_{r \in S} w_{r,ij} M_r\right| \leq L_W\, h\,\alpha\beta,$$

by condition (i) and the definitions of $L_A$, $L_B$ and $L_W$. Each bound holds for the sum of the absolute values of the terms it covers (the codes in a block sum, the $h$ products of two block-sum entries in an entry of $M_r$, the weighted products in an output block), so it also holds for every partial sum formed on the way, in any order. The block sums are therefore exact int8 values, and since integer weights give $L_W \geq 1$, every product and every partial sum formed in the call's multiplications and combinations has magnitude at most $L_W h\alpha\beta$. Condition (ii) keeps this below $2^{31}$, so that int32 accumulation is exact, or, for fp32, below $2^{24}$, where fp32 represents every integer and computes exactly any sum or product whose exact value is such an integer. The call therefore returns its integer product exactly, since its coefficients satisfy (1).

Consider next the calls of one group. Each returns its exact integer product before that is added to the group's running sum, and for a given output entry the calls cover disjoint sets of the group's $g$ inner indices, so every value of the running sum is a sum of at most $g$ terms $A_{tp}B_{pj}$, with $t$ a row of $A$ and $p$ an index of the inner dimension, of magnitude at most $g b_A b_B$. Condition (ii) keeps this within the same exact range, and every group's integer product therefore comes out exact.

In fp32 the integers are held as floating-point numbers, and a product such as $(-1)\cdot 0$ gives $-0$ where the integer is 0; converted as classical int8 converts the integer, it enters the rescaling as $+0$. For the operator we run, any difference between the two representations of a zero group result disappears in any case in the sum over groups, which starts from $+0$, rounds each addition separately and, under IEEE round-to-nearest, never holds $-0$. The realization's output is therefore that of the classical int8 operator at $c$, bit for bit. □

# B CRITERIA, MODELS AND CONTROLS FOR THE SIGN VARIANTS

## B.1 WHAT THE CRITERIA READ, AND WHAT THE OPERANDS DECIDE

Each stability criterion analyzed in Section 3.3 is constant across all sign variants of two-level Strassen. The criteria are computed from the coefficients of (1); write $u_r$, $v_r$ and $w_r$ for the coefficients $u_{r,il}$, $v_{r,lj}$ and $w_{r,ij}$ of product $r$. The measure of expected error of Xie et al. (2026) is $\Phi = \sum_r \|u_r\|_2^2 \, \|v_r\|_2^2 \, \|w_r\|_2^2$; the nonzero coefficients are counted over $U$, $V$ and $W$ together, and the largest coefficient magnitude is taken over all three. With $\alpha_r$ and $\beta_r$ the numbers of nonzero entries of $u_r$ and $v_r$, $a_r = \|u_r\|_1$, $b_r = \|v_r\|_1$, and $\gamma_{ij}$ the number of products with $w_{r,ij} \neq 0$, the prefactor and the stability factor of Ballard et al. (2016, Definitions 4.1 and 4.2) are

$$Q = \max_{i,j}\Big(\gamma_{ij} + \max_{r:\, w_{r,ij}\neq 0}(\alpha_r + \beta_r)\Big), \qquad E = \max_{i,j} \sum_r a_r \, b_r \, |w_{r,ij}|.$$

Table 2 lists four of the 512 sign variants, chosen across the range of their perplexity ratios on Qwen2.5-14B-Instruct. Each criterion takes one value on all four, as on all 512, while the ratio runs from 1.367 to 1056.0. The parameters $Q$ and $E$ of Ballard et al. (2016) read the same coefficient magnitudes and nonzero pattern, so they too take one value on all 512.

Table 2: The stability criteria take one value on every sign variant of two-level Strassen; the measured perplexity ratio does not. Four of the 512 sign variants on Qwen2.5-14B-Instruct under the FP8 schedule of Section 3.1; the ratio is the WikiText-2 perplexity over that of the bf16 model, and the median row is the upper of the two middle ratios. $\Phi$ is the measure of expected error that Xie et al. (2026) compute from the coefficients; the nonzero coefficients are counted over $U$, $V$ and $W$.

| sign variant | $\Phi$ | nonzero coefficients | largest \|coefficient\| | perplexity ratio |
|---|---|---|---|---|
| smallest ratio | 1,024 | 432 | 1 | 1.367 |
| median ratio | 1,024 | 432 | 1 | 1.425 |
| Strassen's original signs | 1,024 | 432 | 1 | 63.11 |
| largest ratio | 1,024 | 432 | 1 | 1056.0 |

Error differences among sign variants depend on the operand distribution. In the models, the schedule of Section 3.1 rounds the block sums of a layer's input to FP8 (e4m3) with one scale for every 128 consecutive entries of a row, and those of its weight, first stored in bf16, with one scale for every $128 \times 128$ block; DeepGEMM multiplies them, returns each product in bf16, and the products are combined in fp32. On synthetic $128 \times 128$ matrices we use a simplified form, one scale per row of each block sum of $A$ and per column of each block sum of $B$, with the products computed in fp32 on a CPU. For every one of the 512 sign variants we draw 200 pairs of operands with independent entries from each of two distributions and average the variant's relative error in the Frobenius norm separately over the even-numbered and the odd-numbered pairs. With entries uniform on $[-1, 1]$ the variants' average errors span a factor of about 1.005, and the two halves' averages correlate across the variants at only 0.121, consistent with the symmetry argument of Section 3.3. With entries uniform on $[0, 1]$ they span a factor of 2.59, and the halves correlate at 0.9997: the variants differ, and differ reproducibly. On such matrices Malik & Becker (2019) also see random signs lower the error, while on Gaussian ones their variants perform about the same. Ballard et al. (2016, Section 5.1) report the same contrast between whole algorithms: among three algorithms of one shape that each minimize one of their criteria, the errors differ clearly on operands uniform on $[0, 1]$ and are practically indistinguishable on operands uniform on $[-1, 1]$.

## B.2 SEVEN MODELS, AND AN OUTLIER TEST

Inside a model the operands are its own activations and weights, so the spread of the sign variants depends on the model. Table 3 repeats the measurement of Section 3.3 on seven models (Yang et al., 2024; 2025; Grattafiori et al., 2024; OLMo et al., 2024; Young et al., 2024). The spread runs from 1.07× on Qwen2.5-7B-Instruct to 772.4× on Qwen2.5-14B-Instruct, while the deployed FP8 kernel's ratio ranges only from 1.0009 to 1.0138.

Since the spread varies this much, one may ask whether a simple property of a model's activations predicts it. A separate test, under the same schedule and on three further models, checked one such

Table 3: The spread of the sign variants of two-level Strassen differs by model, while the deployed FP8 kernel stays close to bf16 on all seven. Perplexity ratios under the FP8 schedule of Section 3.1, each the perplexity of eight 2048-token chunks of WikiText-2 over the bf16 model's on the same chunks. Runs whose producer checksum changed during execution are omitted on Qwen2.5-7B-Instruct and Qwen3-8B. "Variants" counts the sign variants with a valid measurement; the median is the middle value, the upper of the two middle values when the count is even; the spread is the largest ratio over the smallest; "$> 2$" counts the variants whose ratio exceeds 2; the last column is the deployed FP8 kernel's ratio.

| model | variants | smallest | median | largest | spread | $> 2$ | FP8 kernel |
|---|---|---|---|---|---|---|---|
| Qwen2.5-3B | 512 | 1.124 | 1.191 | 68.10 | 60.59 | 41 | 1.0068 |
| Qwen2.5-7B-Instruct | 508 | 1.099 | 1.128 | 1.18 | 1.07 | 0 | 1.0054 |
| Qwen2.5-14B-Instruct | 512 | 1.367 | 1.425 | 1056.0 | 772.4 | 137 | 1.0138 |
| Qwen3-8B | 509 | 1.040 | 1.105 | 6.15 | 5.91 | 14 | 1.0014 |
| Llama-3.1-8B-Instruct | 512 | 1.262 | 1.377 | 2.89 | 2.29 | 5 | 1.0060 |
| OLMo-2-7B-Instruct | 512 | 1.559 | 1.629 | 1.73 | 1.11 | 0 | 1.0023 |
| Yi-6B | 512 | 1.120 | 1.164 | 1.25 | 1.12 | 0 | 1.0009 |

rule: that a model with strong enough activation outliers has a catastrophic sign variant, one whose perplexity exceeds ten times that of its own bf16 model. Its outlier ratio is the largest ratio of the Euclidean norm of a token row to that of the median row, over the inputs of every linear layer in the body of the bf16 model (all but the output layer) and over four 2048-token chunks of WikiText-2. Mistral-7B-v0.1 (Jiang et al., 2023) and Llama-2-7b-hf (Touvron et al., 2023) were chosen before their variants were run, as models whose strong activation outliers should give each a catastrophic variant if the rule held. Their outlier ratios, 1553.4 and 1760.5, exceed the 1057.4 of Qwen2.5-32B-Instruct, yet neither has a catastrophic variant (0 of 512 and 0 of 512), while Qwen2.5-32B-Instruct has 8 of 512. So no threshold on the outlier ratio separates these three models into those above it, with a catastrophic variant, and those below it, without one.

### B.3 A ROTATION, OTHER SUMMATION ORDERS AND REPEATED RUNS

Changing the operands without changing their product moves the spread, and not in one direction. Methods for removing outliers, such as QuaRot and SpinQuant (Ashkboos et al., 2024; Liu et al., 2025), rotate the operands, $A \mapsto AH$ and $B \mapsto H^\top B$ with $H$ orthogonal, which leaves $AB$ unchanged. With the simplified rounding of Appendix B.1, on $128 \times 128$ products with one level of Strassen and 80 draws of the operands, a randomized Hadamard rotation narrows the spread of the average errors (the largest over the smallest) over its 32 stored sign patterns, which repeat some sign variants of Section 2.1, from 2.137 to 1.300, when every column of $A$ carries the same offset, and widens it, from 1.240 to 1.317, when the columns' offsets have random signs and three percent of them are forty times larger than the rest.

With the coefficients and the input fixed, the order in which the 49 products are added is one more choice that leaves the product exact. We ran sixteen realizations of two-level Strassen on Qwen2.5-14B-Instruct, eight sign variants and eight that also permute blocks, each under its original order and seven sampled reorderings of its products; the sign variants of Section 3.3 all keep the original order, so their spread does not come from reordering. A reordering permutes the products throughout, in how they are prepared and launched as well as in how every output block adds them. On one of them the order alone moves the perplexity ratio from 4.6080 to 68.6687, a $14.90\times$ span. On the classical path, reordering the deployed FP8 kernel's scaling groups along the inner dimension, together with their scales, moves its perplexity ratio across eight orders by a factor of only 1.00459.

Within the tested container instance, none of these differences is run-to-run variation. Eight fresh processes on eight GPUs of one container instance return, for each of the two orders at that realization's extremes, exactly the same perplexity ratio (relative difference 0.0). On Qwen2.5-14B-Instruct, running one of four 1,024-token inputs twice, or again after a different input, leaves every logit unchanged under the unrepaired realization, repaired #2 and both row-local controls; for repaired #1, Appendix C.1 reports how its chosen answers reproduce.

The rescore of Section 3.3 that replaces the later text with filler uses 280 windows of 1,024 tokens, alternating WikiText-2 test and C4 validation text, with four scored positions per window and two filler texts per position; its interval resamples the windows.

# C Details of the tests for prefix invariance

## C.1 What the two inputs of a comparison share

The comparisons of Section 3.1 hold everything but the replaced text fixed. Both inputs run in one process with batch size one, through one realization with the same weights and the same order of its products, and they have the same length, so in the fast realizations their token rows are padded with zeros (to a multiple of 16) and split into blocks identically. An answer is scored by summing the log probabilities of its tokens, each read at the position before it; on the original input a single forward pass over the question and the answer gives all of them, and each candidate answer is scored on its own. For the answer token at position $p$, the second input keeps positions 1 to $p-1$, the token's allowed prefix, and replaces position $p$ and every later position with as many new tokens, so the original token is still the one scored, from position $p-1$; an answer's score under replacement therefore adds up scores from separate forward passes, one for each of its tokens. A realization that keeps prefix invariance gives every answer token the same score from both inputs, and so every item the same choice.

The two replacements differ only in where the new tokens come from. The filler is text from the validation split of C4 (Raffel et al., 2019): each item gets one window of it, shared by all its answers and positions, and each replacement takes that window's first tokens. The greedy continuation is generated by the bf16 model from the allowed prefix, one most likely token at a time, for exactly the replaced length; each is generated once for its prefix and length and reused for every realization. OpenBookQA and ARC-Easy each contribute the same first 240 test items in every condition; this subset size follows the compute budget for the separate forward pass required at every scored answer-token position. On the 960 answers of OpenBookQA the continuation built at an answer's first token begins with that token 69 times. The answers changed in Table 1 count the items whose chosen answer differs between the two inputs, including a change from one wrong answer to another, with 95% intervals from resampling the items.

Repeating the comparison reproduces its outcome. For the unrepaired realization, repaired #2 and both row-local controls, repeating an input within one process gives identical logits (Appendix B.3). Across two runs of the whole OpenBookQA comparison in different container instances, every chosen answer of repaired #1 and of both row-local controls was reproduced, although some log probabilities differed.

Table 4 resolves the same comparisons at every scored answer-token position, including positions from items whose chosen answer does not change.

Table 4: Token-level changes on Qwen2.5-14B-Instruct under the replacements of Table 1. At each scored candidate position, $\Delta$ is the original target token's log probability on the original input minus its log probability after replacement; all 3308 OpenBookQA or 4315 ARC-Easy positions are included per realization and replacement. The last three columns summarize $|\Delta|$ in nats; the 90th percentile is the empirical inverse cumulative distribution function. Both row-local controls have 0 changed target scores and identical full logit rows under all three comparisons.

| condition | realization | changed (%) | mean | median | 90th pct. |
|---|---|---|---|---|---|
| OpenBookQA, continuation | repaired #1 | 80.14 | 0.186 | 0.097 | 0.491 |
| | repaired #2 | 89.00 | 0.297 | 0.179 | 0.727 |
| OpenBookQA, filler | repaired #1 | 90.15 | 0.190 | 0.101 | 0.491 |
| | repaired #2 | 99.88 | 0.310 | 0.187 | 0.762 |
| ARC-Easy, filler | repaired #1 | 99.72 | 0.459 | 0.204 | 1.245 |
| | repaired #2 | 99.72 | 0.695 | 0.320 | 1.884 |

**The repair rules.** For the repairs, a token row $x_t$ uses

$$d_t = 2^{\lceil \log_2(\max\{\mathrm{RMS}(x_t), 10^{-3}\}) \rceil},$$

with the root mean square (RMS) taken over the full row: we divide the original row by $d_t$ before the fast mixing and multiply its decoded output row by $d_t$ afterward. For each weight-block-sum entry, we take its bf16 prequantization value and its original DeepGEMM block scale, consider the two e4m3 values at that scale whose decoded values bracket it, and initialize at the nearer one. For the $R$ product terms at the same position, let $e_r$ be the decoded candidate minus the bf16 prequantization block sum and set

$$H_{rs} = \langle w_r, w_s \rangle \langle u_r, u_s \rangle,$$

where $u_r$ and $w_r$ are the coefficient vectors of product $r$ in (1). We sweep $r = 1, \ldots, R$ in order; a candidate change $\delta$ is accepted when its fp32-computed change in $e^\top He$, $2\delta(He)_r + \delta^2 H_{rr}$, is negative. We stop after a sweep with no change and make at most 12 sweeps. This weight-only search uses no activation or text calibration examples. Repaired #1 uses one Strassen level and one classical level; repaired #2 uses two levels of the $\Phi$-best seven-product algorithm of Section 3.1, with the coefficient arrays distributed with the implementation.

### C.2 The 14 changed answers

Table 5 lists every OpenBookQA item whose chosen answer repaired #1 changes when the text after the allowed prefix is replaced by the greedy continuation, and on all 14 of them both row-local controls keep their answer.

### C.3 How a later row reaches an earlier output

Section 3.2 traces the dependence to the block sums of the linear layers, and a single multiplication shows it without a model. We multiply a Gaussian input of 40 rows by a Gaussian $512 \times 512$ weight matrix, padding the input with zero rows to 48 so that two-level Strassen splits its rows into four blocks of 12. The block sums are rounded to FP8 as the models' schedule of Appendix B.1 rounds them, without repairs, and the products are computed in fp32 on the rounded values; a row counts as changed when any of its entries differs at all. We then replace row 40, the last row before the padding, by a fresh Gaussian row and record which output rows change. Under two-level Strassen with Strassen's original signs, and under the algorithm of repaired #2 without its repairs, output rows 4, 16 and 28 change besides row 40 itself, and no other: the rows at the same offset in the other three blocks, all earlier than row 40. With the classical product computed in fp32, from the unrounded operands or from operands rounded to FP8 in the same way, only row 40 changes.

### C.4 Other models

The dependence is not particular to multiple-choice scoring: it changes the next-token predictions of Qwen2.5-14B-Instruct, Qwen2.5-3B, Llama-3.1-8B-Instruct, OLMo-2-7B-Instruct and Qwen3-8B. For each model we take 256 windows of 1,024 tokens, alternating between WikiText-2 and C4, keep each window's first 256 tokens and replace the rest by the corresponding tokens of two other windows, one from each corpus. Each window and replacement form a pair, and we compare the top-1 predictions at positions 1 to 255, whose predicted tokens lie in the kept quarter. Table 6 gives the percentage of these predictions that change for each model and realization; under the two row-local controls together, none of the 1,305,600 predictions compared on the five models changes (0 flips).

### C.5 Other requests

Requests that share a forward pass are exposed in the same way. We split a 1,024-token window into a request A, its first quarter, and a request B, the rest, pack both into one input and build no key–value cache. The position indices restart at B, which the model's implementation reads as two packed sequences, each with a causal attention mask of its own; in a control run with the bf16 model, replacing A leaves every logit of B unchanged, which confirms that the attention is separated. We then replace B by the corresponding tokens of two other windows and compare A's next-token

Table 5: The 14 OpenBookQA items on which repaired #1 changes its chosen answer when the text after the allowed prefix is replaced by the bf16 model's greedy continuation, on Qwen2.5-14B-Instruct, grouped by how the change affects correctness. Each entry gives the item's number in the test split (counted from 0) and its question, then the answer chosen on the original input → the answer chosen under replacement; ✓ marks a chosen answer that is correct, and the correct answer is given on the right when neither is.

| | | |
|---|---|---|
| *Wrong before and after the change* (9) | | *correct answer* |
| 37 | What is used for sensing visual things?<br>*tibia → nerves* | *cornea* |
| 41 | What is different about birth in humans and chickens?<br>*Mother → Fertilization* | *the hard shell* |
| 50 | Some berries may be eaten by<br>*a bear or wolf → a bear or lion* | *a bear or person* |
| 53 | To improve health, what is a good strategy?<br>*restaurant food → business trip* | *a spa trip* |
| 54 | A girl and her mom have the same<br>*date of birth → shirt* | *number of toenails* |
| 84 | Ocean water contains<br>*copious amounts of seltzer → scant amounts of sodium chloride* | *copious amounts of the combination of Na and Cl* |
| 93 | When looking for good soil for plants, typically what is optimal?<br>*compact and hard → dry and sandy* | *malleable and nutritious* |
| 103 | A farmer harvests seeds from some plants, such as tomatoes, in order to plant them later on. These seeds, once planted<br>*have their own sunlight → have their own dirt* | *contain their necessary nutrition* |
| 227 | How is electricity produced from the ocean?<br>*chemical reactions produced from the salt in the water → decaying organic material from sealife* | *energy is accessed underwater from tides* |
| *From correct to wrong* (3) | | |
| 119 | Endangered pandas are sometimes<br>*confined to enclosures to be viewed by the public ✓ → accidentally dropped into volcanoes* | |
| 132 | A desert environment is<br>*arid, parched, and sun-baked ✓ → lush, green, and tropical* | |
| 205 | There are various creatures that live in forests, such as<br>*whitetails ✓ → giant fish* | |
| *From wrong to correct* (2) | | |
| 136 | What is the formula of the substance which best helps plants grow<br>*CO2 → H2O ✓* | |
| 204 | Through DNA, a rabbit will have long ears if<br>*parents were also rabbits → genetic contributors had long ears ✓* | |

distribution at its last position, the token A would emit next. Over the 768 pairs on Qwen2.5-14B-Instruct, OLMo-2-7B-Instruct and Qwen2.5-3B, that distribution changes on every pair (768 of 768, with a total variation above zero) under the unrepaired realization and under repaired #2, and on none under the controls; Table 6 gives, per model, how often its most likely token changes.

On Qwen2.5-3B we also record, over 64 such pairs, the key and value projections of A's rows at ten of the model's 36 layers (layers 0 to 5, 8, 16, 24 and 35). Under the unrepaired realization and repaired #2 every recorded projection differs, in every row of A and on every pair, and the differences already appear at the first layer, before any attention has combined positions; under both controls none differs.

Table 6: The dependence on replaced text, per model. Window test: the percentage of next-token predictions in the kept first quarter of a 1,024-token window that change when the rest of the window is replaced. Packed test: two requests in one forward pass, the number of pairs and the percentage of pairs in which the first request's most likely next token changes when the second request is replaced. The packed test ran on three models, under the unrepaired realization and repaired #2; both row-local controls change no prediction in either test.

| | window test (%) | | | packed test | | |
|---|---|---|---|---|---|---|
| model | unrepaired | repaired #1 | repaired #2 | pairs | unrepaired (%) | repaired #2 (%) |
| Qwen2.5-14B-Instruct | 22.1 | 8.2 | 12.7 | 208 | 18.3 | 13.0 |
| Qwen2.5-3B | 21.3 | 9.9 | 14.6 | 320 | 20.9 | 11.9 |
| Llama-3.1-8B-Instruct | 24.8 | 9.4 | 15.2 | – | – | – |
| OLMo-2-7B-Instruct | 40.9 | 10.0 | 15.0 | 240 | 29.6 | 12.9 |
| Qwen3-8B | 18.2 | 7.8 | 12.7 | – | – | – |

## D CONSTRUCTION DETAILS

**Bitwise checks in the models.** In the verification runs of Section 4.2, the output of two-level Strassen at code bound 31 equals classical int8's bit for bit in all 15,456 invocations of a replaced linear layer on Qwen2.5-14B-Instruct, 36,288 on Llama-3.1-8B-Instruct, 36,288 on OLMo-2-7B-Instruct and 40,824 on Qwen3-8B.

**Quantization specification for the model runs.** The quantizer takes fp32 inputs and computes scales in fp32, grouping 128 consecutive entries along the inner dimension, with a separate activation scale for each token row and weight scale for each output column within a group. For a group $x$, its scale is $d = \max|x|/b$, with $d = 1$ for an all-zero group, and its codes are $x/d$ rounded to nearest with ties to even, with no additional clipping. The inner dimension is zero-padded to a multiple of 128. For each group in ascending inner-dimension order, each entry $P_{tj}$ of the exact integer group product is rescaled as $(P_{tj}d_{A,t})d_{B,j}$ using two separate fp32 multiplications and added to a zero-initialized fp32 output. After the last group the output is cast to the model input dtype, bf16 here, and the bias is then added. The model runs at $b = 31$ and $b = 127$ differ only in the code bound.

**The code bound in context.** Quantized fast convolution meets the same growth in its transformed tiles and handles it by clipping them and retraining (Mori et al., 2024); condition (i) of Proposition 1 instead keeps the codes small enough that no block sum overflows.

**The code bounds of condition (i).** The code bounds $\lfloor 127/L_A \rfloor$ and $\lfloor 127/L_B \rfloor$ that condition (i) admits are sharp, since codes whose signs follow the coefficients reach them; Dumas et al. (2009, Theorem 3.1) prove a sharp bound of this kind for the integer intermediates of Winograd's variant of Strassen's algorithm before reduction modulo a prime. For two-level Strassen the bound 31 makes the quantization step 4.0968 times as coarse as that of full-range int8.

**Other algorithms with 49 multiplications.** Other algorithms with the same number of multiplications pay more than two-level Strassen: each of the 14,236 algorithms with 49 multiplications for $4 \times 4$ blocks released with AlphaTensor (Fawzi et al., 2022) has $L_A \geq 7$, so none admits a code bound above 18.

**Dense uniform codebooks on FP8.** The code bound is set by the need for every block sum to stay exact, and an 8-bit floating-point format meets that need with far fewer evenly spaced codes than int8. In a binary floating-point format with $p$ significant bits, the implicit bit included, a dense uniform codebook $\{0, \pm\delta, \ldots, \pm J\delta\}$ whose every sum of $L$ codes is representable in the same format has $J \leq \lfloor 2^p/L \rfloor$. Such sums include every multiple $q\delta$ with $|q| \leq LJ$, so it suffices that one multiple beyond $2^p$ is not representable: writing $\delta = a \cdot 2^s$ with $a$ odd, $(2^p + 1)\delta$ has the odd part $(2^p+1)a \geq 2^p+1$, which needs more than $p$ significant bits. Hence $LJ \leq 2^p$, and a finite exponent range can only lower the bound. For e4m3, with four exponent bits and three stored significand bits ($p = 4$), and the $L_A = 4$ codes that a block sum of two-level Strassen adds, at most nine evenly

spaced codes remain (for example $-4, \ldots, 4$), against the $2 \cdot 31 + 1 = 63$ codes that condition (i) admits in int8 at the same eight bits.

**The corrected computation's certificate.** At code bound 127, with groups of 128 and blocks of inner length $h = 32$, so that a call spans one group as condition (iii) requires, two-level Strassen with the overflow correction of Section 4.3 computes classical int8 bit for bit, provided its integer arithmetic runs in int32 and the quantization, the rescaling and the order of the fp32 additions across groups are those of classical int8. A block sum of codes, formed exactly in a wider integer type, satisfies $|\hat{A}|, |\hat{B}| \leq 4 \cdot 127 = 508$, which int8 cannot hold, so the unsplit computation violates condition (i) of Proposition 1; the following split instead keeps every multiplicand in int8. With $R_A = \lfloor (\hat{A} + 128)/256 \rfloor$ and $A_0 = \hat{A} - 256 R_A$, and likewise for $\hat{B}$, the parts satisfy $A_0, B_0 \in [-128, 127]$ and $|R_A|, |R_B| \leq 2$, so each of the four products in (2) multiplies int8 values. For one entry of a reconstructed block product over a block of inner length $h$, the absolute values of all the weighted scalar terms of (2) add up to at most $h\,(128 + 256 \cdot 2)^2 = 13{,}107{,}200$. The combination $C_{ij} = \sum_r w_{r,ij} M_r$ of (1) adds at most $L_W = 16$ reconstructed products for an output block, so every partial sum stays below $16 \cdot 13{,}107{,}200 < 2^{31}$ and int32 computes each block of the output exactly, in any order of its integer additions. By (2) and the algorithm's exact identity, each group's integer result is then the classical group product at code bound 127. That product is bounded by $128 \cdot 127^2 = 2{,}064{,}512 < 2^{24}$, so its conversion to fp32 is exact, and the unchanged rescaling and fp32 order give classical int8's output bits. The three layer slices of Appendix E confirm this identity on each of them.

**Model quality.** Table 7 lists the excess values that Figure 4 plots, with the bf16 model's own negative log-likelihood beside them. Table 8 gives ordinary task accuracy on the same first 240 OpenBookQA test items, using the natural inputs and likelihood scoring of Section 3.1.

Table 7: Increase in negative log-likelihood per token over the bf16 model on eight 2048-token chunks of WikiText-2 ($10^{-3}$ nats; lower is better), raw and with every operator and the bf16 model at its own best temperature on the tested grid, chosen for each chunk on the other seven (adjusted). The bf16 NLL is raw. Each certified column is measured on the classical int8 operator its realization computes bit for bit, the one at $b = 127$ with the overflow correction; all three operators use groups of 128 along the inner dimension.

| | bf16 NLL | deployed FP8 kernel | | certified, $b = 31$ | | certified, $b = 127$ | |
|---|---|---|---|---|---|---|---|
| | (nats) | raw | adjusted | raw | adjusted | raw | adjusted |
| Qwen2.5-14B-Instruct | 1.311 | 13.7 | 13.6 | 60.7 | 59.8 | 3.2 | 3.3 |
| Llama-3.1-8B-Instruct | 1.871 | 5.9 | 5.9 | 14.5 | 14.5 | 1.8 | 1.8 |
| OLMo-2-7B-Instruct | 2.078 | 2.3 | 1.9 | 11.0 | 13.3 | −0.4 | −0.1 |
| Qwen3-8B | 2.129 | 1.4 | 2.5 | −5.5 | 18.9 | 0.0 | −0.3 |

Table 8: OpenBookQA accuracy (%) on the first 240 test items. The int8 columns are measured on the classical reference operators that the certified realizations compute, with groups of 128 and the quantization specification above. Each answer is chosen by its summed token log probabilities, with the first candidate selected in an exact tie.

| | | | classical int8 reference | |
|---|---|---|---|---|
| model | bf16 | deployed FP8 | $b = 31$ | $b = 127$ |
| Qwen2.5-14B-Instruct | 34.58 | 33.75 | 35.00 | 34.17 |
| Llama-3.1-8B-Instruct | 35.42 | 35.00 | 35.00 | 35.00 |
| OLMo-2-7B-Instruct | 37.92 | 39.17 | 37.92 | 38.75 |
| Qwen3-8B | 31.67 | 33.75 | 32.08 | 31.67 |

**Qwen3-8B at code bound 31.** On Qwen3-8B the raw excess at code bound 31 is negative; its bf16 model fits this text best at temperature 1.30 on the tested grid.

## E COST DETAILS

**The correction's multiplications.** Section 5 counts the multiplications of two-level Strassen with the overflow correction of Section 4.3 at code bound 127 on one slice from each of three linear layers: the down projection of layer 4 of Qwen2.5-14B-Instruct, the gate projection of layer 2 of Qwen3-8B and the down projection of layer 2 of Qwen2.5-3B, each with 512 token rows and 1024 output channels, at groups of 128 and blocks of inner length 32. On each slice the corrected computation reproduces classical int8 at code bound 127 bit for bit. Every reconstructed product and group output also matches an exact integer-valued float64 reference, with the integer bounds below $2^{53}$. The entrywise bound $\sum_r |w_{r,ij} M_r| < 2^{24}$ keeps every decoded partial sum exact in fp32 on these slices, and the final rescaled outputs have identical bit patterns. Its correction terms, the products with $R_A$ or $R_B$ in (2), are counted in two ways. Counting only the nonzero entries of $R_A$ and $R_B$ gives 0.79–0.82 of classical int8's multiplications in total; counting every row of $R_A$ and column of $R_B$ that holds a nonzero entry, as a dense product over those rows and columns would, gives 1.43–1.71 times as many.

**The timings.** Every timing of Section 5 multiplies two $4096 \times 4096$ matrices on one H20 GPU from prequantized codes and scales. The slowdowns relative to classical int8 at the same code bound and group size compare the fastest configurations found for the two kernels in the respective tile sweeps of one run. Only configurations whose fast and classical outputs have identical bit patterns are timed; the same check must reject a deliberate sign flip in one decoding coefficient. The classical kernel is separately checked against a grouped fp32 reference at relative maximum error below $10^{-5}$. Each configuration receives eight warmup runs followed by 30 CUDA-event measurements, each preceded by zeroing a 96 MB buffer to evict the cache; we report their upper median. In that sweep, one level of Strassen at code bound 63 takes 2.04, 3.17 and 2.09 times as long as classical int8 at groups of 128, 256 and 512, and two levels at code bound 31 take 27.9 times as long at groups of 256; in every configuration a call spans one group, within the conditions of Proposition 1. Part of this cost comes from splitting a call into short block products: when the executor of our fast kernels runs the 64 block products of the classical algorithm, with nothing of the fast algorithm in it, it already takes 2.25 times as long as our classical int8 kernel at groups of 256. Our classical int8 kernel, which applies the group scales as well, takes 563–774 $\mu$s at this shape across the three group sizes, while one call of the cuBLASLt int8 matrix multiplication that PyTorch exposes, which applies none, takes 815 $\mu$s in the same run.

**The order of the products.** The spreads reported here are over the orders we timed at one fixed kernel configuration for each algorithm, for two $4096 \times 4096$ matrices at groups of 256 and code bound 31: the slowest order takes 1.46 times as long as the fastest for two-level Strassen, and 1.97 times for a hybrid of one Strassen level and one classical level. In two further pools, the fastest order could be found without timing. For each algorithm we drew 200 orders absent from the timings behind those spreads, and compiled and timed each, recording the local memory per thread that the compiler allocated to it. The rule, fixed before the orders were drawn, picks the order with the least local memory, which needs only the compilation. In both pools it picked the fastest of the 200: the picked order's time over the pool's best is 1.000 for two-level Strassen and 1.000 for the hybrid. Since every order computes the same bits, such a search cannot change what the model computes; Yang et al. (2026) likewise confine GPU autotuning to configurations that compute the same bits.

## F REPRODUCIBILITY DETAILS

**From stored outputs to printed numbers.** Every result the paper prints as a numeral is generated from the stored output of the program that produced it, by a rule that reads the value from that output or computes it; the 772.4-fold range of Section 3.3, for example, is read from the stored summary of the runs of the sign variants and rounded to one decimal. Table 9 gives, for each family of results, the arithmetic that produced it and the hardware it ran on.

**What holds on any hardware.** Three kinds of result are exact and so hold on any hardware: the coefficient quantities, the multiplication counts, and the certified operator identities (the bitwise equality of the certified construction and classical int8 on the tiles and in every checked invocation inside the models). Proposition 1 guarantees that identity for any implementation whose integer

Table 9: The arithmetic and hardware behind each family of results, with the sections and appendices where the results appear. Outside the layers a row names, the models run in bf16, and likelihoods are computed from their logits in fp32.

| results | where | arithmetic | hardware |
|---|---|---|---|
| Tests of prefix invariance on OpenBookQA and ARC-Easy, windows and packed requests | 3.1, 3.2; C.1, C.2, C.4, C.5 | fast realizations: block sums rounded to FP8 and multiplied by DeepGEMM's FP8 kernel, products combined in fp32; controls: the bf16 model and DeepGEMM's FP8 kernel on the whole product | GPU |
| How a later row reaches an earlier output | 3.2; C.3 | FP8 rounding simulated by casting to FP8 (e4m3) and back; products in fp32 | CPU |
| Sign variants on seven models, four variants rescored with the later text replaced by filler, summation orders and repeated runs | 3.3; B.2, B.3 | block sums rounded to FP8 and multiplied by DeepGEMM's FP8 kernel, products combined in fp32 | GPU |
| The outlier ratio of the counterexample | B.2 | the bf16 model's activations, row norms in fp32 | GPU |
| Stability criteria, code bounds and headroom, and the code bounds of the AlphaTensor algorithms | 3.3, 4.2, 4.3; B.1 | exact arithmetic on the coefficients | CPU |
| Synthetic ensembles and the rotation | B.1, B.3 | FP8 rounding simulated by casting to FP8 (e4m3) and back, products in fp32, against a float64 reference | CPU |
| Certified construction on tiles: exact trials, sign variants, pairs of an earlier and a later row, and the control that breaks condition (iii) | 4.2 | integer products in int64, rescaled in float64 and accumulated in fp32; the FP8 comparison and the control round their block sums in simulated floating point | CPU |
| Checks of replaced layers inside four models: every invocation compared bit for bit with classical int8 | 4.2; D | int8 arithmetic emulated exactly in fp32, with TF32, the GPU's reduced-precision fp32 mode, disabled | GPU |
| The OpenBookQA test on four models and model quality at code bounds 31 and 127 | 4.2, 4.4; D | the classical int8 operator the certified realization computes, emulated exactly in fp32; references: the bf16 model and DeepGEMM's FP8 kernel | GPU |
| The overflow correction on three layer slices: its bitwise check and its multiplication counts | 4.3, 5; D, E | the corrected and the classical computation in fp32, compared bit for bit; multiplications counted over the entries, or the rows and columns, that overflow | CPU |
| Timings, configuration sweeps and product orders | 5; E | int8 Triton kernels with int32 accumulation, timed against our classical int8 kernel, with PyTorch's int8 matrix product as a timing reference | GPU |
| Multiplication counts without the correction | 4.2, 5 | counted from the algorithms | – |

arithmetic is exact and whose quantization, rescaling and accumulation follow the same specification $c$, and the models' fp32 emulation is such an implementation because every integer intermediate stays below $2^{24}$.

**Software.** The available environment records list the following software: the runs of the sign variants used PyTorch 2.11.0 (Paszke et al., 2019) with CUDA 12.8, the repeated runs PyTorch 2.13.0 with CUDA 12.9 and DeepGEMM 0.1.5.post3, the quality measurements PyTorch 2.13.0 with CUDA 13.0, the packed requests transformers 5.3.0 (Wolf et al., 2019), and the timings PyTorch

2.13.0 with CUDA 12.9 and Triton 3.7.1. The timings and the quality measurements record NVIDIA H20 GPUs.

**Models and data.** The experiments on one model use Qwen2.5-14B-Instruct, except the record of key and value projections in Appendix C.5, which uses Qwen2.5-3B. The window test adds Qwen2.5-3B, Llama-3.1-8B-Instruct, OLMo-2-7B-Instruct (the 1124 release) and Qwen3-8B, and the packed requests use Qwen2.5-14B-Instruct, Qwen2.5-3B and OLMo-2-7B-Instruct; the sign variants on seven models add Qwen2.5-7B-Instruct and Yi-6B to the window test's five; Section 4's runs use Qwen2.5-14B-Instruct, Llama-3.1-8B-Instruct, OLMo-2-7B-Instruct and Qwen3-8B; the overflow correction's slices come from Qwen2.5-14B-Instruct, Qwen3-8B and Qwen2.5-3B; and the counterexample involving the outlier ratio uses Qwen2.5-32B-Instruct, Mistral-7B-v0.1 and Llama-2-7b-hf. The multiple-choice tests use the first 240 test items of OpenBookQA and ARC-Easy, with C4 validation text as the filler (Appendix C.1); the window test (Appendix C.4) and the rescore of four variants in Section 3.3 alternate WikiText-2 test and C4 validation text; and the perplexity and quality measurements over whole sequences use 2048-token chunks of the WikiText-2 test set (Appendices B.2 and D).

**Supplementary material.** The anonymous supplementary material holds the code of the methods and experiments and the stored outputs from which the printed numbers are read, with instructions for rerunning the CPU computations of Table 9 and the GPU experiments.