CHANGE THE PRODUCT, KEEP THE PARAMETERS: ASSOCIATIVE ALGEBRA LAYERS FOR TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Fast matrix multiplication algorithms keep the product fixed and search for a cheaper way to evaluate it. We instead ask whether a Transformer's learned projections can use a different, cheaper product altogether. Building on an associative-algebra construction that replaces ordinary matrix multiplication with a sparser interaction table over the same weight blocks, we construct a family with quadratic arithmetic in the matrix dimension when the physical block size remains fixed, and derive finite-shape constraints for GPU execution. The construction is provably optimal for its bilinear rank by the Alder–Strassen bound and can be realized as row-typed rectangular projections compatible with causal masking and KV-cached decoding. We provide an empirical test of this approach by training two approximately 110M-parameter decoder-only Transformer LMs from the same recipe and 12.3B-token budget, differing only in their feed-forward layer: one uses ordinary dense matrix multiplication and the other uses the associative-algebra product. Across four prompt domains, the algebraic model achieves a 6.2–7.8% increase in end-to-end generation throughput, while obtaining lower scores on all three reported downstream metrics. We treat these results as a feasibility and trainability check for the proposed approach at small scale, leaving further investigation to future work.

1 INTRODUCTION

A Transformer repeatedly applies learned linear maps of the form $Y = XW$. After the feature dimensions are divided into groups, this operation becomes a sum of ordinary rectangular block GEMMs. The row–column rule decides which activation block is multiplied by which weight block; the learned entries of W decide the values of those interactions. Most work on fast matrix multiplication treats the row–column rule as fixed and searches for a cheaper evaluation algorithm.

Strassen first separated the product from its algorithm: the exact product of two 2×2 block matrices can be evaluated with seven block multiplications rather than the conventional eight (Strassen, 1969). Algebraic complexity theory and recent automated searches pursue the same direction (Bürgisser et al., 1997; Fawzi et al., 2022; Dupont et al., 2026). The target is still ordinary matrix multiplication; the algorithm is allowed to change.

We ask the reverse question. A hidden representation and its surrounding layers are learned jointly, so must their intermediate product be ordinary matrix multiplication? What is a natural product that is weaker—and cheaper—but still structured enough to use throughout a network? Our proposal keeps the learned weight blocks and changes the rule by which activation and weight blocks interact.

Consider the smallest case, $q = 2$. We first use square block arrays to display the multiplication rule; later, each row of the same rule becomes a rectangular Transformer projection. Write

$$X = \begin{bmatrix} A & U \\ V & D \end{bmatrix}, \qquad W = \begin{bmatrix} W_a & W_u \\ W_v & W_d \end{bmatrix}. \tag{1}$$

All symbols in this display denote compatible matrix blocks, and every juxtaposition below is one ordinary block GEMM. We call the four positions in each array *block slots*. Ordinary block multiplication gives

$$XW = \begin{bmatrix} AW_a + UW_v & AW_u + UW_d \\ VW_a + DW_v & VW_u + DW_d \end{bmatrix}, \tag{2}$$

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.

and therefore uses eight block GEMMs. One idea is simply to remove the two terms that close the off-diagonal cycle:

$$X \star_2 W = \begin{bmatrix} AW_a & AW_u + UW_d \\ VW_a + DW_v & DW_d \end{bmatrix}. \tag{3}$$

Here "remove UW_v" has a literal implementation-level meaning: that GEMM is absent from the fixed rule for every input. The blocks U and W_v remain; in particular, W_v is still learned and appears in DW_v. Likewise, W_u remains in AW_u. Thus all four weight blocks in equation 1 are independent learned parameters, while the new rule uses six block GEMMs instead of eight. The square display lists two row maps at once. In the Transformer realization, positions are assigned one of two row types: a type-1 token supplies the feature groups (A, U) and evaluates the first output row, while a type-2 token supplies (V, D) and evaluates the second. Each position therefore uses one row of the table.

Why remove exactly these two terms? Four matching-slot products would be cheaper but provide only group-wise scaling. We instead seek a rule that retains cross-group interactions and composes across layers. For a fixed weight W, let $\Phi_W(X) := X \star_2 W$. The rule in equation 3 satisfies

$$\Phi_{W_2}(\Phi_{W_1}(X)) = \Phi_{W_1 \star_2 W_2}(X). \tag{4}$$

Thus two compatible linear maps defined by the rule compose into a map of the same form. The block identity $E = \left[\begin{smallmatrix} I & 0 \\ 0 & I \end{smallmatrix}\right]$ satisfies $E \star_2 W = W$ and $X \star_2 E = X$. This is a structural statement about the parameterization: every learned block can change an output for a suitable algebraic input. The Transformer realization exposes the full bank jointly across its row types.

A bilinear multiplication with these two properties is called an *associative unital algebra*. Here the algebra is the fixed table used by the layer: activations supply its first argument, learned weights its second, and $Y = \mu_{\mathcal{A}}(X, W)$. We write $\mathcal{Q}_2$ for the four-slot law in equation 3.

The pattern generalizes to a directed graph: diagonal slots are vertices, retained off-diagonal slots are edges, and edge–edge products vanish. Section 3 gives the construction and proves associativity.

Can the retained rule be evaluated with fewer than six products? Its *bilinear rank* is the smallest number of scalar multiplications in an exact bilinear algorithm. With compatible matrix blocks, the linear forms in each rank-one term become block-linear combinations and each multiplication becomes one block GEMM; actual time also depends on the rectangular shapes. The Alder–Strassen theorem lower-bounds this rank for finite-dimensional associative unital algebras (Alder & Strassen, 1981; Bläser, 2000). For a graph table with m slots and v vertices, its specialized bound is $R \geq 2m - v$, as derived in section 3. The present law has $m = 4$ and $v = 2$, giving $R(\mathcal{Q}_2) \geq 2 \cdot 4 - 2 = 6$. The displayed evaluation uses six products and is therefore optimal for this multiplication table.

The graph construction is a scalable example of the broader algebraic design. Its size can grow with the layer width. With q feature groups, the arithmetic fraction is $(2q - 1)/q^2$; holding the physical GEMM block size fixed while increasing q gives quadratic arithmetic for square operands. The practical question is which group sizes preserve efficient GPU execution. Section 5 derives this scaling and its finite-size constraints, and section 7.1 measures representative Transformer projection shapes.

We lift the slots to rectangular activation and weight blocks and use the resulting row actions as position-wise Transformer projections. Training evaluates all row types in parallel; autoregressive decoding evaluates only the new position's row while retaining the standard KV cache. Sections 4 and 6 give the construction and costs.

The construction raises two empirical questions: can the restricted layers be trained from scratch, and do their lower arithmetic counts improve execution time? We test the recursive $\mathcal{Q}_2^{\otimes 2}$ law in the feed-forward blocks of an approximately 110M-parameter language model, against a parameter-matched dense baseline, after 12.3B training tokens each. Generation throughput improves by 6.2–7.8% across four prompt domains. Scores on GSM8K, IFEval, and MBPP are lower for the algebraic model. These results connect the algebraic savings to an observed quality–throughput trade-off; the quality of larger algebraic constructions remains to be measured.

2 PRODUCTS AS ARCHITECTURAL CHOICES

2.1 A FIXED TABLE, LEARNED WEIGHTS

Let $\mathcal{A}$ be a vector space with m named slots. At the scalar level, a bilinear multiplication table takes the form

$$[\mu_{\mathcal{A}}(x, w)]_k = \sum_{i,j} c_{ij}^k x_i w_j. \tag{5}$$

The coefficients c_{ij}^k specify which activation slot interacts with which weight slot and where the result is accumulated. They are fixed by the architecture; w is learned. Setting a coefficient to zero removes a product from the rule. With compatible matrix blocks, that product is a GEMM. Ordinary $q \times q$ matrix multiplication is one such table with $m = q^2$.

For $\Phi_w(x) = \mu_{\mathcal{A}}(x, w)$, associativity gives

$$\Phi_{w_2} \circ \Phi_{w_1} = \Phi_{\mu_{\mathcal{A}}(w_1, w_2)}. \tag{6}$$

Thus compatible linear maps compose within the same family. A two-sided identity $1_{\mathcal{A}}$ gives $\Phi_{1_{\mathcal{A}}}(x) = x$ and $\Phi_w(1_{\mathcal{A}}) = w$: every weight slot can affect an output. For Transformer projections, a token supplies one row of the table and the complete weight bank is exposed jointly across row types (section 4). Autoregressive causality additionally requires each projection to act independently on token rows (section 6).

2.2 HOW MANY PRODUCTS ARE NECESSARY?

A rank-r bilinear algorithm writes

$$\mu(x, w) = \sum_{\ell=1}^{r} \gamma_\ell \alpha_\ell(x) \beta_\ell(w), \tag{7}$$

where α_ℓ, β_ℓ are linear forms and γ_ℓ are output vectors. The least r is the bilinear rank $R(\mu)$; additions and fixed linear combinations are counted separately (Bürgisser et al., 1997). For compatible block shapes, each scalar multiplication lifts to one GEMM. The graph construction below uses individual blocks, so it also supports unequal rectangular shapes without adding incompatible blocks.

For a finite-dimensional associative unital algebra, Alder–Strassen gives

$$R(\mathcal{A}) \geq 2 \dim \mathcal{A} - t(\mathcal{A}), \tag{8}$$

where $t(\mathcal{A})$ counts its maximal two-sided ideals (Alder & Strassen, 1981; Bläser, 2000). A two-sided ideal is a subspace stable under multiplication from either side; maximal means maximal among proper such subspaces. For our graph products, t will equal the number of vertices.

The bound applies to a specified multiplication law. Coordinatewise multiplication $[\mu(x, w)]_i = x_i w_i$ has rank m and retains only within-slot interactions. We seek a low-rank law that also couples feature groups. Likewise, the conventional matrix-product counts 8 and 64 refer to its block formula; they are not exact ranks, since already $R(M_2) = 7$.

3 GRAPH PRODUCTS WITH EXACT COST

3.1 THE GRAPH SPECIFIES THE PRODUCT

Let $G = (V, E)$ be a finite directed graph without self-loops. Give each vertex a basis element e_i and each directed edge a basis element a_{ij}. Vertices represent diagonal feature slots; edges represent off-diagonal interaction slots. Define $\mathcal{Q}_G$ by the nonzero basis products

$$e_i e_i = e_i, \qquad e_i a_{ij} = a_{ij}, \qquad a_{ij} e_j = a_{ij}. \tag{9}$$

All other basis products vanish. In coordinates, the complete rule is

$$[\mu_{\mathcal{Q}_G}(x, w)]_i = x_i w_i, \tag{10}$$

$$[\mu_{\mathcal{Q}_G}(x, w)]_{ij} = x_i w_{ij} + x_{ij} w_j. \tag{11}$$

Each vertex contributes one product and each edge contributes two. The two-vertex complete graph recovers equation 3: its missing terms multiply two edge slots.

Proposition 3.1 (Structure). *The algebra $\mathcal{Q}_G$ is associative and unital, with dimension $|V| + |E|$ and identity $\sum_i e_i$. The span J of the edge elements is a two-sided ideal satisfying $J^2 = 0$ and $\mathcal{Q}_G/J \cong \mathbb{F}^{|V|}$.*

The proof is given in section H.

Theorem 3.2 (Exact Bilinear Rank). *For every field $\mathbb{F}$,*

$$R(\mathcal{Q}_G) = |V| + 2|E|. \tag{12}$$

The proof is given in section H.

3.2 A Family with Adjustable Size

For the complete directed graph on q vertices, write $\mathcal{Q}_q$ and $A \star_q B := \mu_{\mathcal{Q}_q}(A, B)$. Arrange vertex slots on the diagonal and edge slots off the diagonal. The rule becomes

$$(A \star_q B)_{ii} = A_{ii}B_{ii}, \tag{13}$$

$$(A \star_q B)_{ij} = A_{ii}B_{ij} + A_{ij}B_{jj}, \quad i \neq j. \tag{14}$$

All q^2 slots are retained, and

$$\dim \mathcal{Q}_q = q^2, \qquad R(\mathcal{Q}_q) = 2q^2 - q. \tag{15}$$

The two-group example is the smallest instance. Larger q gives a systematic arithmetic–interaction trade-off, studied in section 5.

Tensor products give another way to build larger algebras. The pretraining pilot uses the tensor square of the two-group law, with 36 explicit products at dimension sixteen. Its exact rank lies in $[28, 36]$; the direct four-group law attains 28. These are distinct interaction rules. Section C gives the comparison and recursive row counts.

4 Transformer Projections

Let $X \in \mathbb{F}^{M\times K}$ and $W \in \mathbb{F}^{K\times N}$. Partition the input and output features into q groups, with widths s_j and r_j, so $K = \sum_j s_j$, $N = \sum_j r_j$, and $W_{jk} \in \mathbb{F}^{s_j \times r_k}$. The block index is $q \times q$; the physical blocks may be rectangular and unequal.

A token $x = (x_1, \ldots, x_q)$ at position p selects a row $i = \tau(p)$ by a fixed deterministic schedule. For example, $\tau(p) = 1 + ((p-1) \bmod q)$ cycles through the row types. Its projection is

$$y_i = x_i W_{ii}, \tag{16}$$

$$y_j = x_i W_{ij} + x_j W_{jj}, \quad j \neq i. \tag{17}$$

Compatible projections compose using the same block product; a diagonal bank of identity maps gives the identity projection.

Proposition 4.1 (Parameters and Rectangular Cost). *The bank stores KN independent scalars. With $D_{\text{diag}} = \sum_j s_j r_j$, a type-$i$ row costs*

$$C_i = D_{\text{diag}} + s_i(N - r_i) \tag{18}$$

MACs. If M_i rows have type i, the batch and training costs are

$$C^{\text{fwd}}_{\text{proj}} = M D_{\text{diag}} + \sum_i M_i s_i (N - r_i), \tag{19}$$

$$C^{\text{train}}_{\text{proj}} = 3C^{\text{fwd}}_{\text{proj}}. \tag{20}$$

For equal feature groups, every row has dense-relative cost

$$\rho_q = \frac{2q-1}{q^2}. \tag{21}$$

Across all types, the weight bank acts injectively.

The gradient formulas and the three-pass count are given in section B; the parameter-coverage argument is immediate from the block support.

A token's local weight gradient reaches all diagonal blocks and its source row of off-diagonal blocks. Other token types update the remaining rows. These are interactions between feature groups within a token; attention couples token positions. More groups reduce active computation while restricting each position's linear map. The resulting map can still have full rank when its diagonal blocks are invertible.

Projection Shapes. A gated FFN uses two $D \times F$ weights and one $F \times D$ weight. For attention, let $H_q = n_q d_h$ and $H_{kv} = n_{kv} d_h$ denote aggregate query and key/value widths. The Q/K/V/O weight shapes are $D \times H_q$, $D \times H_{kv}$, $D \times H_{kv}$, and $H_q \times D$. The same row law applies to each shape. Gates, pointwise activations, normalization, residual connections, and biases retain their usual definitions. The pretraining pilot replaces only the FFN projections.

5 Scaling and Practical Implementation

The algebra size determines how much arithmetic is removed; the physical block sizes determine how efficiently the remaining products run. These are separate choices. A fixed small algebra gives a constant-factor reduction. Allowing its size to grow changes the asymptotic cost.

5.1 Growing the Algebra

Consider square operands of side $n = qb$, using the complete graph algebra on q vertices and ordinary $b \times b$ products inside each slot. The block-valued algebra is $\mathcal{A}_{q,b} = \mathcal{Q}_q \otimes M_b(\mathbb{F})$, of dimension n^2.

Proposition 5.1 (Quadratic Arithmetic at Fixed Block Size)**.** *The direct block algorithm uses*

$$C(n, q) = (2q^2 - q)(n/q)^3 = \left(\frac{2}{q} - \frac{1}{q^2}\right) n^3 \tag{22}$$

MACs. For any fixed positive block size b, choosing $q = n/b$ gives

$$C(n, n/b) = 2bn^2 - b^2 n = \Theta(n^2). \tag{23}$$

Moreover, $R(\mathcal{A}_{q,b}) \geq 2n^2 - q$.

The rank bound follows from Alder–Strassen; see section H.2.

The quadratic order therefore requires no scalar-size leaves: b can remain large enough for a conventional GEMM. The multiplication law changes with q. At fixed q, equation 22 remains cubic; for $q = \Theta(n^\alpha)$, it is $\Theta(n^{3-\alpha})$. Increasing q also restricts each row's feature interactions and changes how often off-diagonal parameters are used. Computational scaling alone does not establish a quality-preserving scaling law.

5.2 Finite Rectangular Shapes

For a projection $X \in \mathbb{F}^{M \times K}$ with $K \times N$ weights and equal groups, the exact active MAC count is

$$C_{\text{proj}}(M, K, N; q) = \frac{2q - 1}{q^2} MKN. \tag{24}$$

The type-specific terms have dimensions approximately $(M/q) \times (K/q) \times (N/q)$. If efficient leaf GEMMs require at least m_0, k_0, n_0 along these axes, a useful screening condition is

$$q \;\leq\; \min\{M/m_0,\; K/k_0,\; N/n_0\}. \tag{25}$$

The admissible q must also divide the feature widths for equal groups. The criterion provides an initial size filter. For example, requiring all three axes to be at least 128 permits $q \leq 16$ at $M = K = N = 2048$, and $q \leq 32$ at side 4096. Decode requires a separate matrix–vector implementation because its row dimension is small. At non-aligned sizes, the executed tiles include masked rows and columns; their arithmetic can substantially exceed the active count in equation 24.

Algorithm 1 Batched or one-token algebraic projection.

Require: Rows X, positions p_b, weights W, support $\mathcal{B}$
$Y \leftarrow 0$
for each row type i **do**
  $I_i \leftarrow \{b : \tau(p_b) = i\}$
  for each output group j and $k \in \mathcal{B}(i, j)$ **do**
    $Y_j[I_i] \leftarrow Y_j[I_i] + X_k[I_i]W_{kj}$
  end for
end for
return Y

Let $P_d, P_a(q)$ be the effective dense and algebraic compute rates, $\eta_q = P_a(q)/P_d$, and let δ_q be additional overhead divided by dense time. In a compute-dominated model, speedup requires

$$\rho_q/\eta_q + \delta_q < 1. \tag{26}$$

Smaller blocks can reduce η_q enough to offset the arithmetic saving. Ignoring overhead, $q > 2/\eta_q$ is sufficient; along the fixed-b family this reads $n > 2b/\eta_q$. The effective rate must be measured at the resulting leaf shapes. Memory traffic remains a constraint because a batch exposing all row types uses the full weight bank.

5.3 Kernels

We extend the endpoint-product schedules of our existing block-algebra kernels to arbitrary q and rectangular feature widths. One forward kernel accumulates both endpoint products before writing the output. A second schedule computes the diagonal maps over all rows, then the type-dependent maps; this improves reuse when each type has few rows. A dedicated decode kernel performs vector reductions. The activation and weight gradients have separate kernels with one owner per output element.

These kernels consume the canonical weight bank. Tile selection is calibrated on each shape and evaluated with separate timing trials. The next section reports where the arithmetic reduction produces a measured benefit. Each q defines a different architecture. A deployed model keeps q fixed across training, prefill, and decoding; kernel tiles and schedules can adapt to the workload.

6 Training and Inference

Training evaluates all token rows in parallel; autoregressive decoding evaluates one new row per sequence. Both regimes use the same weights and the same position-type schedule.

6.1 One Row Action in Both Regimes

For a support set $\mathcal{B}(i, j)$ determined by the algebra, a row of type i computes

$$y_j = \sum_{k \in \mathcal{B}(i,j)} x_k W_{kj}. \tag{27}$$

For the complete graph construction, $\mathcal{B}(i, j) = \{i, j\}$ as a set. The recursive pilot uses the endpoint sets in section C. Algorithm 1 applies to both.

6.2 Training

Each term in Algorithm 1 is a rectangular GEMM. The two backward maps use the same support, so matrix-product MACs for forward plus backward equal three forward evaluations (proposition 4.1; section B gives the direct-law gradients). For a gated MLP with widths D, F, the three projections have shapes $D \times F, D \times F, F \times D$. On B token rows, their training cost falls from $9BDF$ to $9\rho BDF$. Activations, gates, normalization, and residual operations are additional.

Proposition 6.1 (Causality)**.** *A fixed position-type schedule and row-local algebraic projections, combined with the usual triangular attention mask, preserve autoregressive causality.*

The proof is the standard induction over layers; see section H.3.

6.3 One-Token Inference

At position p, the decoder appends its key/value to the cache, attends to allowed positions, and applies the output and FFN projections. The FFN uses Algorithm 1 with row type $\tau(p)$; an absolute-position schedule keeps this type consistent across training, prefill, and decoding.

For N cached positions, per-head width d_h, and n_{kv} key/value heads,

$$K^{\text{cache}}, V^{\text{cache}} \in \mathbb{R}^{N \times n_{kv} \times d_h}. \tag{28}$$

Storage is $2NH_{kv}$ scalars, with $H_{kv} = n_{kv}d_h$. Let $H_q = n_q d_h$ and let $g = 2$ for a gated MLP ($g = 1$ for a two-map MLP). With algebraic FFN projections and dense attention, one decoder block costs

$$C_{\text{dec}} = 2DH_q + 2DH_{kv} + \rho(g+1)DF + 2NH_q \tag{29}$$

MACs in projections and attention contractions. The dense baseline sets $\rho = 1$. In the tested configuration, $H_q = H_{kv} = D$, $F = 2D$, and $\rho = 9/16$. Its decoder-block projection fraction is

$$\frac{4D^2 + (9/16)\,6D^2}{10D^2} = \frac{59}{80}.$$

Attention contractions and the language-model head further dilute the FFN saving. These counts exclude memory traffic and elementwise operations.

Implementation and Extensions. The smaller products can use off-the-shelf GEMMs; specialized kernels can fuse their products and accumulations. The pilot uses CUDA-graph decode paths for both models. Graph capture reduces launch overhead, while single-token computation can still be limited by memory bandwidth. Section G develops the separate extension to algebraic attention scores, including its cache and normalization formulas; the experiment uses dense attention.

7 Experiments

7.1 Kernel Scaling on Transformer Shapes

We measure both FFN projection directions at the public dimensions of Qwen3-1.7B, 4B, 8B, a Qwen3-30B-A3B expert, and a DeepSeek-V3 expert (Qwen Team, 2025; DeepSeek-AI, 2024). The corresponding (D, F) pairs are $(2048, 6144)$, $(2560, 9728)$, $(4096, 12288)$, $(2048, 768)$, and $(7168, 2048)$. An expert's M counts tokens already routed to that expert. We vary $M \in \{1, 32, 512, 2048\}$ and $q \in \{4, 8, 16, 32, 64\}$.

The study uses synthetic BF16 tensors on one H200 NVL, FP32 accumulation, CUDA Graph replay for both implementations, and preallocated outputs. The primary comparison is ordinary XW against the algebraic projection, without activation or FFN fusion. Calibration selects the faster available cuBLAS/cuBLASLt path for the dense shape and the schedule and tile for the algebraic kernel; separate alternating trials provide the reported medians. Forward and both gradient maps are checked against independent formulas.

Warm replay and a separate cache-flushed measurement characterize different reuse regimes. The reported speedups are kernel-level projection/FFN speedups, not end-to-end Transformer speedups; whole-model throughput, MoE dispatch, and inter-device communication require separate measurements. Fully realizing the acceleration potential will also require fused kernels; we leave such kernel fusion to future work. The additional gated-block comparison merges gate/up on both sides and uses the same activation kernel; its results are reported in the appendix. The extended Qwen3-32B projection sweep and fixed-block square-product measurements are reported in section F. The main-table results already show the dependence of useful subdivision on workload size and physical block shape.

7.2 Small Pretraining Study

We train two approximately 110M-parameter decoder-only language models from scratch on 12.3B tokens each, with one run per model. Both use 12 pre-norm blocks, width $D = 768$, 32 attention

Table 1: Pure expansion projections, BF16 on H200 NVL. The selected q_* has the lowest warm Q-kernel median in the tested grid: $\{4, 8, 16, 32, 64\}$ for the first five shapes and $\{32, 64, 128, 256\}$ for Qwen3-32B. Block dimensions $(m, k, n) = (M/q_*, K/q_*, N/q_*)$ show where subdivision stops, before GPU-tile padding. Times are microseconds; speedups are kernel speedups (dense kernel / algebraic kernel), reported for warm / cache-flushed runs. † marks the largest tested q.

Model shape	M	q_*	Block (m, k, n)	Dense μs	Q μs	Speedup
Qwen3-1.7B	2048	32	$(64, 64, 192)$	75.7	24.9	$3.04\times$ / $2.84\times$
Qwen3-4B	2048	32	$(64, 80, 304)$	160.6	51.1	$3.14\times$ / $2.94\times$
Qwen3-8B	2048	32	$(64, 128, 384)$	320.7	63.6	$5.04\times$ / $4.52\times$
Qwen3-30B-A3B expert	2048	16	$(128, 128, 48)$	13.2	8.4	$1.57\times$ / $1.30\times$
DeepSeek-V3 expert	2048	16	$(128, 448, 128)$	82.3	28.3	$2.91\times$ / $2.47\times$
Qwen3-32B	2048	32	$(64, 160, 800)$	849.2	167.8	$5.06\times$ / $4.86\times$
Qwen3-32B	8192	128	$(64, 40, 200)$	3337.4	400.7	$8.33\times$ / $8.60\times$
Qwen3-32B	32768	$256^\dagger$	$(128, 20, 100)$	13458.6	1169.4	$11.51\times$ / $11.70\times$

Table 2: Measured generation throughput. Gain is the relative increase over the dense model within each domain.

Domain	Dense tok/s	Algebraic tok/s	Gain
Code	987.8	1054.5	6.8%
Math	991.4	1060.6	7.0%
QA	958.6	1033.1	7.8%
WMT ru–en	985.9	1047.1	6.2%

heads, learned positional embeddings, context length 1536, and the GPT-2 tokenizer with vocabulary size 50,257. The bias-free SwiGLU FFN has width $F = 1536$ and three weight matrices of shapes $D \times F, D \times F, F \times D$. The dense model uses ordinary projections; the algebraic model uses the recursive $\mathcal{Q}_2^{\otimes 2}$ action equation 27. Attention, embeddings, and the language-model head are dense in both.

Both runs use two NVIDIA H100 GPUs with a per-GPU batch size of 40, AdamW with peak learning rate 3×10^{-4}, 500 warmup steps, and cosine decay to 10% of the peak. Prompt and response fields from the same public instruction-data mixture are concatenated for training. Full hyperparameters and the reported corpus manifest are in sections D and I. The algebraic FFN's projection MAC fraction is $9/16$.

Generation Throughput. Both models use CUDA-graph single-token decode paths. We measure end-to-end generation throughput including tokenization and sampling across four prompt domains (table 2). The algebraic model has higher throughput in every domain, by 6.2–7.8%. The unweighted mean of the four throughput ratios is 1.070.

Generated lengths differ between models: for example, the mean code response is 195.5 tokens for dense and 182.4 for algebraic. The measurements therefore describe the realized generation workloads. Prompt and response lengths are reported in tables 5 and 6.

Downstream Quality. We use the evaluation harness (Gao et al., 2023) to evaluate GSM8K (Cobbe et al., 2021), IFEval (Zhou et al., 2023), and MBPP (Austin et al., 2021) on the same accelerated inference paths. GSM8K uses five-shot prompting; IFEval and MBPP use zero-shot evaluation. Table 3 reports a concise comparison; all six metric values and available standard errors are in table 8.

The algebraic model scores lower on all three reported metrics. The GSM8K gap is 0.98 points, loose instance-level IFEval gap is 3.4 points; the MBPP pass@1 gap is 2.8 points. The pilot establishes that the recursive layer can be trained and used for faster generation under this recipe, with a measurable quality cost.

Table 3: Downstream scores in percent. Higher is better. The appendix reports the complete metric set and evaluation standard errors.

Task	Metric	Dense	Algebraic
GSM8K	EM, flexible	3.18	2.20
IFEval	Instance, loose	34.5	31.1
MBPP	pass@1	6.60	3.80

8 Related Work

Fast matrix multiplication. Strassen showed that classical matrix multiplication is arithmetically suboptimal (Strassen, 1969); tensor rank and algebraic complexity provide the framework for subsequent bounds (Alder & Strassen, 1981; Bürgisser et al., 1997; Bläser, 2000). Recent work improves the matrix-multiplication exponent while retaining the same target product (Dupont et al., 2026). We instead change the product.

Alternative algebras and structured layers. AlgebraNets and algebraic neural networks replace or generalize ordinary multiplication through alternative or finite-dimensional algebra laws (Hoffmann et al., 2020; Parada-Mayorga & Ribeiro, 2021; Niemczynowicz & Kycia, 2026). Hypercomplex multiplication, Tensor Train, Monarch, and Block Tensor-Train methods impose structure primarily through parameter factorization or reduction (Zhang et al., 2021; Oseledets, 2011; Novikov et al., 2015; Dao et al., 2022a; Qiu et al., 2024). Our setting keeps exactly $d_{\text{in}}d_{\text{out}}$ learned scalars and changes only the multiplication table, selected by certified scalar bilinear rank.

Other efficient Transformers. Matmul-free models remove dense matrix multiplications through alternative representations (Zhu et al., 2024); MoE models use content-dependent expert selection (Fedus et al., 2022). Our method uses fixed type-dependent sparsity with the full weight bank. Dao et al. (2022b) show that memory traffic also affects wall-clock attention time; our runtime experiment keeps attention dense.

9 Discussion and Limitations

The experiment tests one recursive FFN law, one model scale, and one training run per model. Larger algebras, algebraic attention, larger models, and repeated seeds remain untested. Projection benchmarks use synthetic tensors; evaluation standard errors quantify task-example uncertainty, not training-run variation.

The throughput gain is smaller than the FFN arithmetic reduction because attention, the LM head, sampling, tokenization, and memory traffic also contribute. Generation lengths differ, so fixed-length repeated full-model timings are needed to isolate the layer effect.

Equal parameter storage permits different function classes, while associativity guarantees composition rather than quality preservation. Thus the results do not establish an advantage at equal quality or training time.

10 Conclusion

Associative algebras make the multiplication law an architectural choice. A growing graph family retains the dense-sized weight bank and reduces square-product arithmetic to quadratic order at a fixed physical block size.

The kernel measurements identify this arithmetic–latency trade-off on representative Transformer shapes. The pretraining pilot separately shows a generation-throughput gain accompanied by lower downstream scores.

AI USE STATEMENT

We used generative AI tools for spellchecking and grammar checking, for LaTeX-related formatting, and for drafting parts of the paper. All AI-assisted text and formatting were reviewed by the authors, who take full responsibility for the final content, claims, and presentation of this work.

REFERENCES

A. Alder and Volker Strassen. On the algorithmic complexity of associative algebras. *Theoretical Computer Science*, 15:201–211, 1981. doi: 10.1016/0304-3975(81)90070-0.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. arXiv preprint arXiv:2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

Markus Bläser. Lower bounds for the bilinear complexity of associative algebras. *Computational Complexity*, 9:73–112, 2000. doi: 10.1007/PL00001605.

Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1997. doi: 10.1007/978-3-662-03338-8.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Tri Dao, Beidi Chen, Nimit S. Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4690–4721, 2022a.

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

DeepSeek-AI. DeepSeek-V3 technical report. arXiv preprint arXiv:2412.19437, 2024. URL https://arxiv.org/abs/2412.19437.

Emilien Dupont, Marvin Eisenberger, Borislav Kozlovskii, Abbas Mehrabian, Francisco J. R. Ruiz, Abigail See, Renfei Zhou, Josh Alman, Virginia Vassilevska Williams, and Matej Balog. Improving the matrix multiplication exponent with modern optimization and AlphaEvolve. arXiv preprint arXiv:2608.16884, 2026. URL https://arxiv.org/abs/2608.16884.

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*, 610:47–53, 2022. doi: 10.1038/s41586-022-05172-4.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. Zenodo, 2023. URL https://zenodo.org/records/10256836.

Jordan Hoffmann, Simon Schmitt, Simon Osindero, Karen Simonyan, and Erich Elsen. AlgebraNets. arXiv preprint arXiv:2006.07360, 2020. URL https://arxiv.org/abs/2006.07360.

Agnieszka Niemczynowicz and Radosław Antoni Kycia. Fully tensorial approach to hypercomplex-valued neural networks. *Information Sciences*, 755:123796, 2026. doi: 10.1016/j.ins.2026.123796.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, volume 28, pp. 442–450, 2015.

I. V. Oseledets. Tensor-Train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. doi: 10.1137/090752286.

Alejandro Parada-Mayorga and Alejandro Ribeiro. Algebraic neural networks: Stability to deformations. *IEEE Transactions on Signal Processing*, 69:3351–3366, 2021. doi: 10.1109/TSP.2021.3084537.

Shikai Qiu, Andres Potapczynski, Marc Anton Finzi, Micah Goldblum, and Andrew Gordon Wilson. Compute better spent: Replacing dense layers with structured matrices. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 41698–41716, 2024.

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

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

Aston Zhang, Yi Tay, Shuai Zhang, Alvin Chan, Anh Tuan Luu, Siu Cheung Hui, and Jie Fu. Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with $1/n$ parameters. In *International Conference on Learning Representations*, 2021.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. arXiv preprint arXiv:2311.07911, 2023. URL https://arxiv.org/abs/2311.07911.

Rui-Jie Zhu, Yu Zhang, Ethan Sifferman, Tyler Sheaves, Yiqiao Wang, Dustin Richmond, Peng Zhou, and Jason K. Eshraghian. Scalable MatMul-free language modeling. arXiv preprint arXiv:2406.02528, 2024. URL https://arxiv.org/abs/2406.02528.

A Direct Associativity Check

Write an element of $\mathcal{Q}_q$ as (d, n) with $d \in \mathbb{F}^q$ diagonal and n off-diagonal. The product is

$$(d, n) \star_q (d', n') = (dd',\, dn' + nd'), \tag{30}$$

where left and right diagonal multiplication act on the source and target of each arrow; explicitly, $(dn')_{ij} = d_i n'_{ij}$ and $(nd')_{ij} = n_{ij} d'_j$. Since the left and right actions commute, both parenthesizations have diagonal part $dd'd''$. Let

$$\Xi = dd'n'' + dn'd'' + nd'd''.$$

Expanding either parenthesization gives

$$((d, n) \star_q (d', n')) \star_q (d'', n'') = (dd'd'', \Xi),$$
$$(d, n) \star_q ((d', n') \star_q (d'', n'')) = (dd'd'', \Xi),$$

so the law is associative. The element $(\mathbf{1}, 0)$ is the unit. This calculation complements the basis-element argument in proposition 3.1.

B Vector–Jacobian Products

For one token of type i, let $\bar{y}_j$ denote the gradient of the loss with respect to output group j. The typed projection equation 16–equation 17 gives

$$\bar{x}_i \mathrel{+}= \bar{y}_i W_{ii}^\top + \sum_{j \neq i} \bar{y}_j W_{ij}^\top, \tag{31}$$

$$\bar{x}_j \mathrel{+}= \bar{y}_j W_{jj}^\top, \qquad j \neq i, \tag{32}$$

$$\bar{W}_{ii} \mathrel{+}= x_i^\top \bar{y}_i, \tag{33}$$

$$\bar{W}_{ij} \mathrel{+}= x_i^\top \bar{y}_j, \qquad j \neq i, \tag{34}$$

$$\bar{W}_{jj} \mathrel{+}= x_j^\top \bar{y}_j, \qquad j \neq i. \tag{35}$$

Summing these expressions over token types yields gradients for every block. The displayed vector–Jacobian products preserve exactly the block support of the forward multiplication table. Hence dX and dW each have the same matrix-product MAC count as the forward projection, which proves equation 20.

C The Tensor-Square Candidate

Let $J = \mathrm{rad}(\mathcal{Q}_2)$. Although $\mathcal{Q}_4$ and $\mathcal{Q}_2^{\otimes 2}$ are both sixteen-dimensional and have semisimple quotient $\mathbb{F}^4$, they are not isomorphic. In $\mathcal{Q}_4$, the radical satisfies $\mathrm{rad}(\mathcal{Q}_4)^2 = 0$. In the tensor square,

$$\mathrm{rad}(\mathcal{Q}_2^{\otimes 2}) = J \otimes \mathcal{Q}_2 + \mathcal{Q}_2 \otimes J,$$

since the right-hand side is nilpotent and the quotient is $(\mathcal{Q}_2/J) \otimes (\mathcal{Q}_2/J) \cong \mathbb{F}^4$. Its square contains $J \otimes J \neq 0$ because $J\mathcal{Q}_2 = \mathcal{Q}_2 J = J$. The tensor-square law therefore permits some two-step radical interactions that strict $\mathcal{Q}_4$ removes. The 36-versus-28 comparison concerns two non-isomorphic products on the same number of coordinates.

C.1 Rectangular Counts for Recursive Products

For $\mathcal{A}_L = \mathcal{Q}_2^{\otimes L}$, types and feature-group indices are bit strings in $\{0,1\}^L$. A token of type α uses intermediate groups β whose coordinates come from the corresponding endpoints α, δ. With input widths s_β and output widths r_δ,

$$C_\alpha^{(L)} = \sum_{\delta} \sum_{\substack{\beta:\, \beta_k \in \{\alpha_k, \delta_k\} \\ k=1,\dots,L}} s_\beta r_\delta. \tag{36}$$

Table 4: Training configuration shared by both models.

Quantity	Value
Decoder blocks	12
Model width	768
FFN width	1536
Attention heads	32
Context length	1536
Vocabulary size	50,257
Parameters	approximately 110M
Training tokens	12.3B
Hardware	2 NVIDIA H100 GPUs
Microbatch size	40 per GPU
Gradient accumulation	1
Maximum optimizer steps	100,000
Warmup steps	500
Peak learning rate	3×10^{-4}
Final learning-rate fraction	0.1
Weight decay	0.1
Adam (β_1, β_2)	$(0.9, 0.95)$
Adam ϵ	10^{-8}
Gradient clipping norm	1.0

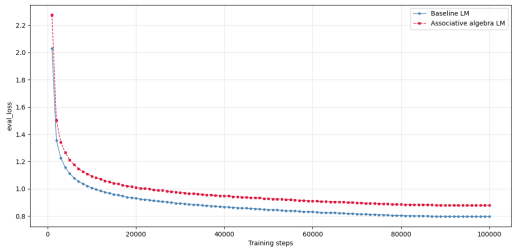


Figure 1: Evaluation loss curves for the baseline and associative algebra LMs.

For uniform types, a block (β, δ) occurs with probability $2^{-h(\beta,\delta)}$, where h is the Hamming distance. Hence

$$\bar{C}^{(L)} = \sum_{\beta,\delta} 2^{-h(\beta,\delta)} s_\beta r_\delta. \tag{37}$$

Equal group widths give ratio $(3/4)^L$ relative to a dense projection. At $L = 2$, the full product uses $4 \cdot 1 + 8 \cdot 2 + 4 \cdot 4 = 36$ block products. More generally the recursive algorithm uses 6^L products, establishing $R(\mathcal{Q}_2^{\otimes L}) \leq 6^L$.

D Experimental Details

D.1 Training Configuration

Both models are trained from scratch using the same instruction-data mixture described in Appendix I, tokenizer, initialization scheme, and optimization recipe. Each model is trained for 100,000 optimizer steps on two NVIDIA H100 GPUs, with a per-GPU batch size of 40 and sequence length 1536. This corresponds to 12.288B token slots, or approximately 12.3B consumed training tokens.

Both models contain 110,631,168 trainable parameters.

The software versions are CUDA 13.0, PyTorch 2.13.0, and `transformers` 4.54.1. The tokenizer is GPT-2 BPE; positional embeddings are learned. FFN projections are bias-free.

Both models use BF16 autocast for forward and backward computation, while trainable parameters are stored in FP32. Evaluation loss curves for both models are shown in Figure 1.

Table 5: Generation speed measurements for the baseline model. Each domain contains 100 unique prompts. The first of four measurement runs was excluded as a warm-up; the three retained runs are shown in the *Measurements* column.

Domain	Mean prompt tokens	Mean response tokens	Measurements tok/sec	Mean speed tok/sec	95% CI margin of error
Code	42.9	195.5	987.5, 988.2, 987.6	987.8	1.0
Math	54.3	223.8	990.1, 993.7, 990.3	991.4	5.0
QA	17.1	92.6	954.0, 962.8, 959.0	958.6	11.0
WMT ru–en	263.8	190.0	988.0, 986.0, 983.7	985.9	5.2

Table 6: Generation speed measurements for the associative algebra model. Each domain contains 100 unique prompts. The first of four measurement runs was excluded as a warm-up; the three retained runs are shown in the *Measurements* column.

Domain	Mean prompt tokens	Mean response tokens	Measurements tok/sec	Mean speed tok/sec	95% CI margin of error
Code	42.9	182.4	1055.5, 1053.7, 1054.2	1054.5	2.2
Math	54.3	227.8	1058.9, 1061.9, 1060.9	1060.6	3.8
QA	17.1	110.8	1031.9, 1034.6, 1032.6	1033.1	3.5
WMT ru–en	263.8	204.9	1047.3, 1048.0, 1046.0	1047.1	2.6

D.2 Generation Speed Measurement Protocol

Generation speed was measured separately on four domains: *code* (code generation), *math* (solving mathematical problems), *qa* (general questions), and *wmt ru–en* (Russian-to-English translation). Each domain contains 100 unique prompts.

Responses were generated with sampling using temperature $= 0.8$, top-$p = 0.95$, and top-$k = 50$, with a maximum of 256 generated tokens. All generation was performed in BF16 precision. Measurements were conducted four times for each model and domain. The first run was excluded from the reported averages to reduce the effect of CUDA and GPU-cache warm-up; the remaining three measurements were used for the reported means.

The margin of error for each mean was computed using the Student's t-distribution with a 95% confidence interval. The reported generation speed includes the complete measured generation workload, including tokenization and sampling.

D.3 Downstream Tasks Evaluation Protocol

Downstream evaluation was performed using `lm-eval-harness==0.4.9.1` on the generative tasks GSM8K, MBPP, and IFEval. The maximum number of generated tokens was 512, the batch size was 1, and generation was performed in BF16 precision. All other generation parameters were left at their default values for the `lm_eval.simple_evaluate()` call.

D.4 Complete Downstream Results

Table 8 reports the complete set of supplied downstream results on their original $[0, 1]$ scale. The standard errors are those reported by the evaluation harness; "N/A" indicates that no standard error was supplied.

The underlying evaluation outputs were 0.0318423 and 0.0219864 for GSM8K, 0.066 and 0.038 for MBPP, and 0.345324 and 0.310552 for IFEval, respectively. The corresponding reported standard errors were 0.004836 and 0.004039 for GSM8K and 0.011115 and 0.008559 for MBPP; no standard error was supplied for IFEval.

Table 7: Associative algebra model generation speed relative to the baseline model.

Domain	Relative speedup (%)
Code	6.8
Math	7.0
QA	7.8
WMT ru–en	6.2

Table 8: Complete downstream results on the original $[0, 1]$ scale. EM denotes exact match.

Task	Metric	Dense	Algebraic
GSM8K	EM, flexible-extract	0.0318 ± 0.0048	0.0220 ± 0.0040
MBPP	pass@1	0.0660 ± 0.0111	0.0380 ± 0.0086
IFEval	Instruction-level loose accuracy	0.3453	0.3106

D.5 Reproducibility Status

Both models were trained with the same model architecture, optimization hyperparameters, data pipeline, and training budget, with the dense and associative algebra models differing in the multiplication law used by the FFN projections. The training run uses two NVIDIA H100 GPUs and 100,000 optimizer steps with a per-GPU batch size of 40, corresponding to approximately 12.3B consumed token slots.

The available GPT-2 pipeline uses two recursive levels, with feature splits $768 \to 384 \to 192$. For zero-based position p, its four-valued row type is $\lfloor (p \bmod 768)/192 \rfloor$, corresponding to the two-bit index α. This schedule assigns consecutive intervals to each type. The projection microbenchmarks use cyclic types to cover growing group counts uniformly. The $9/16$ arithmetic fraction holds for the stated tensor-square row action with equal groups, irrespective of the type frequencies.

The generation measurements use four domains with 100 unique prompts per domain, four measurement runs, and three retained runs after warm-up exclusion. Margins of error are based on Student's t-distribution at a 95% confidence level. Downstream evaluation uses `lm-eval-harness==0.4.9.1` with the settings specified above.

E Kernel Benchmark Details

The primary benchmark isolates the product. The dense control selects the faster cuBLAS/cuBLASLt preference during calibration. The additional SwiGLU test applies identical gate/up merging and activation code to both methods. All tensors are synthetic BF16 with FP32 accumulation. Timings use H200 NVL, PyTorch 2.13.0+cu130, and Triton 3.7.1. For each measured configuration, twelve held-out trials alternate the order of the two methods. Warm trials repeatedly replay the same graph. In the flushed protocol, a 512 MiB buffer is touched before each single replay, outside timing. The protocols also differ in sustained duty cycle; they characterize two reuse regimes. At $M = 1$, warm replay repeats one row type. Full decoder timings require its actual sequence of types and cache accesses. Small-shape verification passed 86 forward/backward checks, including non-power-of-two group counts, with maximum relative L_2 error 0.00208. Chosen forward tiles and sampled large-shape gradient entries are checked independently. Algebraic gradient kernels use a fixed $(32, 64, 32)$ tile with four warps; only forward schedules are calibrated. The complete FFN is compared against a PyTorch composition. Concurrent compute processes are checked before and after each measurement. An interrupted preliminary screen is excluded. Performance counters were unavailable under the server permissions, so the report makes no counter-based bottleneck attribution. MoE measurements exclude dispatch and communication. DeepSeek dimensions are evaluated in BF16; its production FP8 implementation is outside this comparison.

Table 9: Complete SwiGLU forward at fixed $q = 32$, BF16 on H200 NVL. Both implementations merge gate/up. Times are warm-replay medians in microseconds. The last column reports the separately measured cache-flushed speedup. These are isolated FFNs on public model shapes.

Model shape	D	F	M	Dense μs	Algebra μs	Speedup (warm / flushed)
Qwen3-1.7B	2048	6144	1	30.2	8.7	3.45× / 2.46×
Qwen3-1.7B	2048	6144	2048	273.7	107.4	2.55× / 2.45×
Qwen3-4B	2560	9728	1	49.1	10.4	4.74× / 3.21×
Qwen3-4B	2560	9728	2048	539.3	193.8	2.78× / 2.68×
Qwen3-8B	4096	12288	1	83.6	12.6	6.64× / 4.12×
Qwen3-8B	4096	12288	2048	1031.9	247.3	4.17× / 3.92×
Qwen3-30B-A3B expert	2048	768	1	11.5	6.7	1.71× / 1.42×
Qwen3-30B-A3B expert	2048	768	2048	40.3	23.2	1.74× / 1.54×
DeepSeek-V3 expert	7168	2048	1	33.4	8.5	3.94× / 2.75×
DeepSeek-V3 expert	7168	2048	2048	304.2	81.3	3.74× / 3.40×

Table 10: Forward plus activation and weight gradients, $M = 2048$, $q = 32$, warm replay. The dense library preference is calibrated separately for each of its three GEMMs. Times are microseconds.

Model shape	Direction	Dense	Algebra	Speedup
Qwen3-1.7B	up	225.6	177.0	1.27×
Qwen3-1.7B	down	223.8	127.0	1.76×
Qwen3-4B	up	455.9	341.6	1.33×
Qwen3-4B	down	465.9	280.2	1.66×
Qwen3-8B	up	912.0	465.0	1.96×
Qwen3-8B	down	954.0	372.6	2.56×
Qwen3-30B-A3B expert	up	40.0	61.4	0.65×
Qwen3-30B-A3B expert	down	38.5	67.6	0.57×
DeepSeek-V3 expert	up	260.8	146.1	1.79×
DeepSeek-V3 expert	down	275.9	205.0	1.35×

F Larger Pure Products

The extended experiment keeps the pure-product comparison and synthetic BF16 inputs. It tests both Qwen3-32B FFN projection directions at $M \in \{2048, 8192, 32768\}$ and $q \in \{32, 64, 128, 256\}$, and square products with $n \in \{4096, 8192, 16384, 32768\}$ and fixed block sizes $b \in \{128, 256\}$. The public Qwen3-32B configuration gives hidden width 5120 and intermediate width 25600. No model weights are loaded.

Calibration tests eight tile configurations for each of the fused and two-stage Q schedules and selects the dense cuBLAS/cuBLASLt preference. Each calibration uses three short timing trials; twelve separate evaluation trials alternate the two methods. Warm trials use up to 40 graph replays, choosing a repeat count that targets at least 2 ms for the faster operation when possible. Cache-flushed trials use one replay after touching 512 MiB outside the timed interval. Large tensors are allocated once per shape. Only the forward product is measured in this extension.

Every candidate is checked against independent FP32 sums at at least 2048 sampled output entries, covering diagonal and off-diagonal terms and the full row range. The largest sampled relative L_2 error is 0.00196. Additional $q = 128, 256$ checks exercise nonzero type offsets and row counts not divisible by q. The larger-size grid contains 32 configurations, each measured under both cache protocols.

At fixed b, each operand and the output contain n^2 entries, while the number of active block products is $2q^2 - q$. The hardware measurements test the finite-size behavior of this family. They do not establish a quality-preserving model scaling law; q changes between members of the family.

Across the tested eightfold increase in n, the $b = 128$ Q latency has endpoint log–log slope 2.01. This is a description of these four measured sizes, separate from the exact quadratic arithmetic bound in proposition 5.1.

Table 11: Pure $D \to F$ projection: optimized dense GEMM versus the algebraic product at fixed $q = 32$, BF16 on H200 NVL. Times are warm-replay medians in microseconds. The last column gives warm / cache-flushed speedup. No activation or FFN fusion is included.

Model shape	D	F	M	Dense μs	Algebra μs	Speedup (warm / flushed)
Qwen3-1.7B	2048	6144	1	9.5	3.8	2.51× / 1.89×
Qwen3-1.7B	2048	6144	2048	75.7	24.9	3.04× / 2.84×
Qwen3-4B	2560	9728	1	14.0	4.1	3.43× / 2.72×
Qwen3-4B	2560	9728	2048	160.6	51.1	3.14× / 2.94×
Qwen3-8B	4096	12288	1	30.7	4.4	7.07× / 4.14×
Qwen3-8B	4096	12288	2048	320.7	63.6	5.04× / 4.52×
Qwen3-30B-A3B expert	2048	768	1	6.5	3.3	2.00× / 1.36×
Qwen3-30B-A3B expert	2048	768	2048	13.4	10.5	1.28× / 1.14×
DeepSeek-V3 expert	7168	2048	1	10.9	3.8	2.87× / 2.20×
DeepSeek-V3 expert	7168	2048	2048	82.4	28.5	2.89× / 2.51×

Table 12: Qwen3-32B projection shapes at four subdivision sizes. Block dimensions $(m, k, n) = (M/q, K/q, N/q)$ precede GPU-tile padding. Times are warm medians in milliseconds; bold Q times are the lowest among the four tested q values for that shape. The last column gives warm / cache-flushed speedup.

$K \to N$	M	q	Block (m, k, n)	Dense ms	Q ms	Speedup
5120 → 25600	2048	32	(64, 160, 800)	0.849	**0.168**	5.06× / 4.86×
		64	(32, 80, 400)	0.829	0.232	3.58× / 3.75×
		128	(16, 40, 200)	0.779	0.248	3.15× / 3.25×
		256	(8, 20, 100)	0.789	0.266	2.97× / 3.19×
5120 → 25600	8192	32	(256, 160, 800)	3.290	0.583	5.65× / 5.88×
		64	(128, 80, 400)	3.290	0.523	6.30× / 6.68×
		128	(64, 40, 200)	3.337	**0.401**	8.33× / 8.60×
		256	(32, 20, 100)	3.316	0.414	8.00× / 8.10×
5120 → 25600	32768	32	(1024, 160, 800)	13.446	2.335	5.76× / 5.76×
		64	(512, 80, 400)	13.336	2.055	6.49× / 6.83×
		128	(256, 40, 200)	13.596	1.558	8.73× / 8.68×
		256	(128, 20, 100)	13.459	**1.169**	11.51× / 11.70×
25600 → 5120	2048	32	(64, 800, 160)	0.810	0.181	4.49× / 4.27×
		64	(32, 400, 80)	0.731	**0.147**	4.97× / 4.85×
		128	(16, 200, 40)	0.813	0.174	4.66× / 4.41×
		256	(8, 100, 20)	0.778	0.243	3.20× / 3.32×
25600 → 5120	8192	32	(256, 800, 160)	3.339	0.576	5.79× / 5.95×
		64	(128, 400, 80)	3.354	**0.396**	8.48× / 8.30×
		128	(64, 200, 40)	3.312	0.411	8.06× / 8.07×
		256	(32, 100, 20)	3.294	0.552	5.96× / 6.38×
25600 → 5120	32768	32	(1024, 800, 160)	13.564	2.354	5.76× / 5.72×
		64	(512, 400, 80)	13.509	**1.519**	8.89× / 8.92×
		128	(256, 200, 40)	13.356	1.684	7.93× / 7.97×
		256	(128, 100, 20)	13.248	1.958	6.76× / 6.84×

G Attention Extension and Detailed Cost Accounting

An autoregressive Transformer uses the same trained layer in two computational regimes. Teacher-forced training evaluates all token rows and all allowed query–key pairs in parallel under the triangular mask. Inference first processes the prompt in parallel and then contributes one new row per active sequence while reusing the KV cache. The algebra, weights, and position types remain fixed across both regimes.

The two algorithms use three properties of the construction. Each projection acts on one token row; the type $\tau(p)$ is determined by the position; and a pair score depends only on the current query, one

Table 13: Fixed-block-size square products. Subdivision stops at $b \times b$ blocks; algebra size grows as $q = n/b$. Warm medians are milliseconds; bold Q times are the lower of the two tested block sizes for that n. The final column includes the separate cache-flushed measurement.

n	b	q	Dense ms	Q ms	Speedup (warm / flushed)
4096	128	32	0.210	**0.044**	4.81× / 4.47×
4096	256	16	0.205	0.058	3.56× / 3.19×
8192	128	64	1.773	**0.166**	10.69× / 10.63×
8192	256	32	1.729	0.225	7.70× / 7.42×
16384	128	128	13.881	**0.689**	20.15× / 19.95×
16384	256	64	13.999	0.991	14.13× / 14.08×
32768	128	256	114.265	**2.868**	39.85× / 38.85×
32768	256	128	117.518	5.344	21.99× / 21.58×

Algorithm 2 Parallel training with algebraic attention.

Require: Hidden rows H_b, absolute positions p_b, weight banks
1: $i_b \leftarrow \tau(p_b)$ for every row b
2: Compute Q, K, V with Algorithm 1
3: **for** each allowed query–key pair (b, u) **do**
4: Compute s_{bu} using equation 39–equation 41
5: **end for**
6: $P_{bu} \leftarrow \mathrm{softmax}_{u:\, p_u \le p_b}(s_{bu})$
7: $O_b \leftarrow \sum_{u:\, p_u \le p_b} P_{bu} V_u$
8: Apply the typed output and feed-forward projections
9: Compute next-token loss and backpropagate

key, and their two types. Hence a decoder computes each K_u, V_u pair once and reuses it for every future query. The triangular mask determines the temporal dependency, while the algebra selects feature groups inside each admitted pair.

G.1 Training

For one attention head, let $d_h = qm$ and split $Q_p, K_u \in \mathbb{R}^{d_h}$ into q groups of width m. For types $i = \tau(p)$ and $j = \tau(u)$, define

$$F(i,j) = \begin{cases} \{i\}, & i = j, \\ \{i,j\}, & i \neq j, \end{cases} \tag{38}$$

The algebra determines the unnormalized score

$$r_{pu} = \sum_{a \in F(i,j)} \left\langle Q_p^{(a)}, K_u^{(a)} \right\rangle. \tag{39}$$

Its support contains

$$k_{ij} = |F(i,j)|m \tag{40}$$

scalar coordinate products: m when $i = j$ and $2m$ otherwise. To derive a default scale, suppose at initialization that the selected query and key coordinates are independent, centered, and have unit variance. Then $\mathrm{Var}(r_{pu}) = k_{ij}$. The variance-matched score is therefore

$$s_{pu} = \frac{r_{pu}}{\sqrt{k_{ij}}}. \tag{41}$$

Attention applies this scaling to logits before softmax. In ordinary dense attention the score uses all d_h coordinates, so $k_{ij} = d_h$ and equation 41 reduces to the standard $1/\sqrt{d_h}$ rule. The algebraic score uses only m or $2m$ coordinates, giving $1/\sqrt{m}$ or $1/\sqrt{2m}$. Before normalization, equation 39 is precisely the corresponding block of $Q \star_q K^\top$:

$$S_{ii} = Q_{ii} K_{ii}^\top, \qquad S_{ij} = Q_{ii} K_{ji}^\top + Q_{ij} K_{jj}^\top \quad (i \neq j).$$

Algorithm 3 Cached decoding with algebraic attention.

Require: New row h_p, full cache $(K_u, V_u)_{u<p}$, weight banks
$i \leftarrow \tau(p)$
Compute Q_p, K_p, V_p with Algorithm 1
Append full K_p, V_p to the cache
for each allowed cached position $u \le p$ **do**
  $j \leftarrow \tau(u)$
  Compute s_{pu} using equation 39–equation 41
end for
$P_p \leftarrow \mathrm{softmax}(s_p)$; $O_p \leftarrow \sum_{u \le p} P_{pu} V_u$
Apply the typed output and feed-forward projections
return updated hidden row and KV cache

For $\mathcal{Q}_2^{\otimes 2}$, types are bit strings $\alpha, \delta \in \{0,1\}^2$ and the selected groups are

$$F_{\otimes}(\alpha, \delta) = \{\beta : \beta_k \in \{\alpha_k, \delta_k\},\ k = 1, 2\}. \tag{42}$$

A pair at Hamming distance $h(\alpha, \delta)$ uses 2^h groups, hence $k_{\alpha\delta} = 2^h m$ and scale $1/\sqrt{2^h m}$.

We use the support-size normalization in equation 41. Algebra groups lie within each attention head, and every rotary-embedding pair stays within one group.

The temporal mask and value aggregation are

$$\widetilde{s}_{pu} = \begin{cases} s_{pu}, & u \le p, \\ -\infty, & u > p, \end{cases} \tag{43}$$

$$P_{pu} = \mathrm{softmax}_u(\widetilde{s}_{pu}), \qquad O_p = \sum_{u \le p} P_{pu} V_u. \tag{44}$$

The first design uses the algebraic score together with dense PV.

Proposition G.1 (Causality)**.** *Suppose the hidden state at position p depends only on input positions at most p. Algebraic projections, the score in equation 41, and the masked aggregation in equation 43–equation 44 preserve this property.*

Proof. Equations equation 16–equation 17 act independently on each token row. At query position p, the mask admits keys and values at $u \le p$. The product selects feature groups within an admitted pair and therefore preserves the temporal dependency graph. □

For projections, dX and dW repeat the forward block support, giving the three-pass count in equation 20. For equal groups the resulting projection fractions are $\rho_2 = 3/4$, $\rho_4 = 7/16$, and $9/16$ for the recursive $\mathcal{Q}_2^{\otimes 2}$ construction.

For $\mathcal{Q}_q$ with equal group widths, let M be the number of allowed query–key pairs and $M_=$ the number with equal endpoint types. The exact score fraction is

$$\rho_{\mathrm{score}} = \frac{2M - M_=}{qM}. \tag{45}$$

For balanced long sequences, $\rho_{\mathrm{score}} \to \rho_q$. Counting only the attention contractions, structured $QK^\top$ with dense PV has the arithmetic speedup ceilings

$$S_{\mathrm{attn,fwd}} = \frac{2}{1 + \rho_{\mathrm{score}}}, \qquad S_{\mathrm{attn,train}} = \frac{7}{3 + 4\rho_{\mathrm{score}}}, \tag{46}$$

where the training count uses score recomputation in the backward pass and includes dQ and dK. Forward plus backward then contains four structured score contractions—forward $QK^\top$, score recomputation, dQ, and dK—and three dense value contractions—PV, dP, and dV.

Table 14: Balanced-sequence arithmetic ceilings for structured $QK^\top$ and dense PV.

Law	ρ	Forward	Training
$\mathcal{Q}_2^{\otimes 2}$	$9/16$	$1.280\times$	$1.333\times$
$\mathcal{Q}_4$	$7/16$	$1.391\times$	$1.474\times$

Let $\mathcal{P}$ be the learned projections in one decoder block. If projection ℓ sees $B_{\ell i}$ rows of type i and $B_\ell = \sum_i B_{\ell i}$, set

$$C^{\mathrm{fwd}}_{\mathrm{proj,dense}} = \sum_{\ell\in\mathcal{P}} B_\ell d_\ell^{\mathrm{in}} d_\ell^{\mathrm{out}},$$

$$C^{\mathrm{fwd}}_{\mathrm{proj,alg}} = \sum_{\ell\in\mathcal{P}} \left[B_\ell D_{\mathrm{diag},\ell} + \sum_i B_{\ell i} s_{\ell i} (d_\ell^{\mathrm{out}} - r_{\ell i}) \right]. \tag{47}$$

Under the standard MHA/GQA convention with aggregate query width $H_q = n_q d_h$ and M allowed attention pairs, the projection and attention-contraction forward MAC counts are

$$C^{\mathrm{fwd}}_{\mathrm{dense}} = C^{\mathrm{fwd}}_{\mathrm{proj,dense}} + 2MH_q, \tag{48}$$

$$C^{\mathrm{fwd}}_{\mathrm{alg}} = C^{\mathrm{fwd}}_{\mathrm{proj,alg}} + (1+\rho_{\mathrm{score}})MH_q, \tag{49}$$

and forward plus backward gives

$$C^{\mathrm{train}}_{\mathrm{dense}} \simeq 3C^{\mathrm{fwd}}_{\mathrm{proj,dense}} + 7MH_q, \tag{50}$$

$$C^{\mathrm{train}}_{\mathrm{alg}} \simeq 3C^{\mathrm{fwd}}_{\mathrm{proj,alg}} + (3+4\rho_{\mathrm{score}})MH_q. \tag{51}$$

GQA changes the K, V projection widths and cache storage through H_{kv}; every query head still performs its own score and value contraction, which gives the H_q factor above.

G.2 Inference

Prompt prefill uses the parallel computation above and initializes the cache. At each layer and for each sequence, the cache after position p contains

$$K^{\mathrm{cache}}, V^{\mathrm{cache}} \in \mathbb{R}^{N_p \times n_{kv} \times d_h}, \tag{52}$$

where N_p is the number of retained positions. Thus the storage is $2N_pH_{kv}$ scalars, exactly as in the dense Transformer. Keys and values keep all d_h coordinates; the type of entry u is recovered as $\tau(u)$ from its position.

Algorithm 3 gives the one-row update. Under GQA, each query head uses its assigned key/value head. Each cached key and value is computed once and reused by every later position. Different future query types may select different groups of the same cached key, which is why the cache stores the complete key vector. Past queries and hidden rows do not enter the update.

For example, with $q = 4$, a type-3 query reads group 3 from a cached type-3 key and groups $\{3, 1\}$ from a cached type-1 key. A later type-2 query reads groups $\{2, 1\}$ from that same type-1 key. The full cached key supports both queries without recomputation.

Writing $D_{\mathrm{diag},\ell} = \sum_a s_{\ell,a} r_{\ell,a}$, applying equation 18 to every projection $\ell \in \mathcal{P}$ gives

$$C^{(i),\mathrm{dec}}_{\mathrm{proj,alg}} = \sum_{\ell\in\mathcal{P}} \left[D_{\mathrm{diag},\ell} + s_{\ell,i}\big(d_\ell^{\mathrm{out}} - r_{\ell,i}\big) \right], \tag{53}$$

$$C^{\mathrm{dec}}_{\mathrm{proj,dense}} = \sum_{\ell\in\mathcal{P}} d_\ell^{\mathrm{in}} d_\ell^{\mathrm{out}}. \tag{54}$$

Unequal group widths make the algebraic cost position-type dependent.

For a standard decoder block, let D be the model width, F the MLP width, and $g = 1$ for a two-projection MLP or $g = 2$ for a gated MLP with two $D \to F$ maps. The dense one-token projection cost is

$$C^{\mathrm{dec}}_{\mathrm{proj,dense}} = 2DH_q + 2DH_{kv} + (g+1)DF. \tag{55}$$

Table 15: One-token decode at one layer. Projection entries assume equal algebra groups; attention entries hold for the current cache composition.

Quantity	Dense	$\mathcal{Q}_q$
KV cache scalars	$2N_pH_{kv}$	$2N_pH_{kv}$
Projection MACs	$C^{\mathrm{dec}}_{\mathrm{proj,dense}}$	$\rho_q C^{\mathrm{dec}}_{\mathrm{proj,dense}}$
Score MACs	N_pH_q	$\rho^{(i)}_{\mathrm{score,dec}} N_pH_q$
Value MACs	N_pH_q	N_pH_q

With equal algebra groups and all these projections replaced, every token type has the same cost

$$C^{\mathrm{dec}}_{\mathrm{proj,alg}} = \rho_q C^{\mathrm{dec}}_{\mathrm{proj,dense}}, \qquad \rho_q = \frac{2q-1}{q^2}. \tag{56}$$

After appending K_p and V_p, let the cache contain N_p allowed keys, with n_j keys of type j and $\sum_j n_j = N_p$. A type-i query head uses one group for its n_i matching keys and two groups for the others. Its exact score cost is

$$C^{(i)}_{QK,\mathrm{head}} = m(2N_p - n_i). \tag{57}$$

Across n_q query heads,

$$C^{(i)}_{QK,\mathrm{alg}} = \frac{H_q}{q}(2N_p - n_i), \qquad \rho^{(i)}_{\mathrm{score,dec}} = \frac{2N_p - n_i}{qN_p}. \tag{58}$$

Dense value aggregation costs N_pH_q. The one-token projection and attention-contraction MAC counts are therefore

$$C^{(i),\mathrm{dec}}_{\mathrm{dense}} = C^{\mathrm{dec}}_{\mathrm{proj,dense}} + 2N_pH_q, \tag{59}$$

$$C^{(i),\mathrm{dec}}_{\mathrm{alg}} = C^{(i),\mathrm{dec}}_{\mathrm{proj,alg}} + \left(1 + \rho^{(i)}_{\mathrm{score,dec}}\right) N_pH_q. \tag{60}$$

For a balanced cache, $n_i = N_p/q$ and $\rho^{(i)}_{\mathrm{score,dec}} = \rho_q$. For example, $\mathcal{Q}_4$ reduces the score contraction to $7N_pH_q/16$. Dense value aggregation still costs N_pH_q, so attention as a whole costs $23/32$ of dense attention, an arithmetic speedup of $32/23 \approx 1.39$. Decode time remains linear in N_pH_q and cache storage remains linear in N_pH_{kv}; the changed product improves the constants in the projections and score contraction, while dense value aggregation sets the attention floor.

For $\mathcal{Q}_2^{\otimes 2}$, a query type α has exact per-head score cost

$$C^{(\alpha)}_{QK,\mathrm{head}} = m \sum_{\delta \in \{0,1\}^2} 2^{h(\alpha,\delta)} n_\delta. \tag{61}$$

A balanced cache gives score fraction $9/16$ and attention-only speedup 1.28.

Decode exposes skinny GEMMs or GEMVs and can be limited by weight and cache bandwidth. Under GQA, the cache stores $2N_pH_{kv}$ key/value scalars, while the arithmetic scales with H_q because every query head has its own scores and probabilities. A lower arithmetic count therefore need not produce the same relative speedup as in training. Timing this extension requires per-token latency and throughput measurements across batch sizes and cache lengths.

H Graph-Algebra and Scaling Proofs

H.1 Graph Algebra

Proof of proposition 3.1. A triple of basis elements with at least two edges vanishes under either bracketing. With one edge, both bracketings retain that edge exactly when the adjacent vertices match its source and target. With no edges, both require matching vertices. Bilinearity proves associativity. The sum $\sum_i e_i$ acts as an identity on every basis element. Multiplying an edge from either side returns an edge or zero, so its span is a two-sided ideal and $J^2 = 0$. Removing that span leaves the independent products $e_ie_i = e_i$, giving $\mathcal{Q}_G/J \cong \mathbb{F}^{|V|}$. □

Proof of theorem 3.2. Equations equation 10–equation 11 give the upper bound $R(\mathcal{Q}_G) \leq |V|+2|E|$. Every maximal two-sided ideal contains J: its image in the semisimple quotient cannot contain a nonzero nilpotent ideal. Maximal ideals therefore correspond to the $|V|$ maximal ideals of $\mathbb{F}^{|V|}$, so $t(\mathcal{Q}_G) = |V|$. The Alder–Strassen bound then gives

$$R(\mathcal{Q}_G) \geq 2(|V| + |E|) - |V| = |V| + 2|E|,$$

which matches the upper bound. □

H.2 Quadratic Scaling

Proof of proposition 5.1. There are $2q^2 - q$ ordinary block products, each costing b^3 MACs. Since $n = qb$,

$$(2q^2 - q)b^3 = \left(\frac{2}{q} - \frac{1}{q^2}\right) n^3 = 2bn^2 - b^2 n.$$

The edge ideal tensored with $M_b(\mathbb{F})$ is nilpotent, while the quotient is a direct product of q copies of $M_b(\mathbb{F})$. Hence $t(\mathcal{A}_{q,b}) = q$, and Alder–Strassen gives

$$R(\mathcal{A}_{q,b}) \geq 2n^2 - q.$$

□

H.3 Causality

Proof of proposition 6.1. The row action depends only on the features and type at the current position. At position p, triangular attention admits keys and values only at positions $u \leq p$. Induction over layers therefore preserves dependence only on positions at most p. □

I TRAINING DATA SOURCES

The training and evaluation data comprise a mixture of open-source datasets available on Hugging Face [1]. Table 16 reports the number of records imported from each source dataset. The training split includes only the training splits of the original datasets.

Deduplication. To avoid unintended duplication from source datasets that combine other datasets, we retain only the first occurrence of each `query–response` pair, where duplicates are defined by an exact character-by-character match.

Decontamination and junk filtering. For each raw sample, the `query` field is checked for near-duplicates against a pool of benchmark queries from MBPP, HumanEval, MMLU, GSM8K, HellaSwag, and PIQA. Similarity is measured using Jaccard similarity over 3-character shingles, with MinHash + LSH used for efficient candidate generation followed by an exact Jaccard recheck. For samples with `domain == "coding"`, every ```` ```python ... ``` ```` fenced code snippet in the `response` field is additionally compared against MBPP reference solutions in the `code` field. This detects cases in which the query has been paraphrased or rewritten but the response remains close to a benchmark solution. Samples with Jaccard similarity above `DECONTAM_JACCARD_THRESHOLD=0.80` are removed.

Trivially empty, too-short, and degenerate samples are also removed using domain-specific checks: code-specific checks for `domain == "coding"` and a generic repetition check otherwise.

Table 16: Imported records by dataset, as recorded for the training mixture.

Dataset	Records
GSAI-ML/ReFusion	2881860
openbmb/UltraData-SFT-2605	2651052
MBZUAI/LaMini-instruction	2540109
OLMo-Coding/starcoder-python-instruct	1087321
nvidia/Nemotron-Post-Training-Dataset-v2	1145725
nvidia/Nemotron-RL-knowledge-mcqa	611930
nvidia/Nemotron-SFT-Instruction-Following-Chat-v2	604155
ZeroAgency/ru-big-russian-dataset	441537
t-tech/T-Wix	438000
meta-math/MetaMathQA	381136
OpenCoder-LLM/opc-sft-stage2	280319
jtatman/python-code-dataset-500k	207994
NTU-NLP-sg/xCodeEval	204545
d0rj/orca-math-word-problems-200k-ru	197488
Helsinki-NLP/opus-100	165138
sayhan/strix-philosophy-qa	133760
open-r1/OpenThoughts-114k-math	88843
allenai/tulu-3-sft-personas-math-filtered	80535
openbmb/UltraInteract_sft	75781
Post-training-Data-Flywheel/AutoIF-instruct-61k	61345
argilla/ifeval-like-data	56054
camel-ai/math	49079
MERA-evaluation/MERA	48771
bingbangboom/philosophia-QA	47394
allenai/tulu-3-sft-personas-math-grade-filtered	47042
MuskumPillerum/General-Knowledge	34877
Vikhrmodels/GrandMaster-PRO-MAX	29123
AITISPEC/physics-russian	19780
attn-signs/russian-code	19148
MexIvanov/CodeExercise-Python-27k-ru	18923
ajibawa-2023/Python-Code-23k-ShareGPT	17092
teknium/OpenHermes-2.5	14649
RushabhShah122000/python-expert-dataset	12567

[1] `https://huggingface.co/datasets`

Dataset (continued)	Records
qwedsacf/competition_math	11613
microsoft/NextCoderDataset	9319
mizinovmv/ru_ifeval-like-data	3822
attn-signs/russian-easy-instructions	2101
newfacade/LeetCodeDataset	2082
greengerong/leetcode	2050
jondurbin/airoboros-3.2	1710
ise-uiuc/Magicoder-Evol-Instruct-110K	656
nvidia/Nemotron-SFT-ARC-AGI-v1	521
LLiserginov/russian-instructions-10k	178
HuggingFaceH4/ifeval-like-data	3