Transformers as Generic Deep Neural Networks: Exact Compilation, the Cost of Removing Activation–Activation Products, and Multi-Agent Attention Clustering

Sergey Salishev[1,*], Zeev Volkovich[2], and Oleg Granichin[1]

[1]St. Petersburg State University, Saint Petersburg 199034, Russia
[2]Department of Software Engineering, Braude College of Engineering, Karmiel 2161002, Israel
[*]Corresponding author: s.salischev@spbu.ru

Supplementary Information

Supplementary Note: Tightness and scope of the bound

Squares only. With $\Phi = \{(\cdot)^2\}$ and one nonlinear layer, PW is within a factor two of the bilinear rank of the matrix multiplication tensor, since polarization turns each product into two squares and each square is one product. Fast matrix multiplication [1, 2] then gives strict graphs with $\mathrm{PW} = O(n^{2.373})$ for $n \times n$ products, so no $\Omega(N^2 d)$ bound holds for this Φ, and the factor $d + 2d/N$ between the strict graph of Theorem 1 and Theorem 2 is not forced. Conversely, on the set of Lemma 8 a graph over $\{(\cdot)^2\}$ with PW squarings is an arithmetic circuit with PW nonscalar multiplications, and nonscalar multiplications never exceed the total arithmetic cost [3]. For $d = N$, a lower bound of order $N^{2+\delta}$ for this Φ would therefore prove that the exponent of matrix multiplication is at least $2 + \delta$, an open problem. For fixed d, $N^2 d = O(N^2)$ and the theorem is tight in order. MAC arrays and tensor units run the classical product, and the strict graph follows it.

Unrestricted Φ. On a bounded box, Score is computed by a product-free graph with $N^2 + 2Nd$ unary evaluations, so no argument that ignores the regularity of Φ can improve Theorem 2 beyond the additive $2Nd$. Map the $n = 2Nd$ inputs affinely into $[0, 1)$. For $p = 1, \ldots, n$ let ψ_p send $x \in [0, 1)$, written in binary with infinitely many zero digits, to the number whose digit at position $rn + p$ is the r-th digit of x and whose other digits are zero. The affine sum $z = \sum_p \psi_p(x_p)$ interleaves the digits of all inputs, has infinitely many zero digits, and so determines every x_p. One further unary map per output, $\phi_{ij}(z) = S_{ij}$, decodes the score. The total is $n + N^2$ evaluations. The construction reaches the bound by first encoding its inputs; one evaluation of a linear form alone cannot compute a product on an open set, because $\phi(a \cdot x + b)$ is constant on hyperplanes.

Transform-wise scope. Applied to a whole block as one map, the nonlinear rank is at most the number of outputs, $Nhd = 3.78 \times 10^4$ for DeiT-Tiny, below the pointwise work of the source block. A compilation that never forms $QK^\top$ is therefore outside Theorem 2. The theorem

concerns the attention contractions as transforms, which is the form in which every Transformer implementation and every NPU operator set presents them.

Supplementary Note: NPU operator support

Supplementary Table S1 answers, for ten edge accelerators and from their documented operator support, whether the dense strict, Hadamard and extended graphs run entirely on the NPU. "Whole" means the documentation lists every operation of the graph as supported on the accelerator; otherwise the cell lists the operations left to the host CPU or GPU, or, on R-Car, to the on-chip DSP. MatMul is a matrix product of two activations, as in $QK^\top$ and PV, and Rsqrt is the inverse square root. Where the compiler or runtime is open source we read the placement from its code at the commit given in the reference; for the three closed toolchains, Rockchip, Jetson DLA and Edge TPU, we use the vendor documentation. A square runs as the elementwise product $x \odot x$, which every accelerator in the table supports. A softmax counts as on the NPU when the NPU runs it as a native row-wise operator or runs both the exponential and the reciprocal. Where an accelerator lacks GELU, the MLP uses $x\,\sigma(1.702x)$, which every accelerator in the table runs with its sigmoid and its elementwise product; edge deployments also replace GELU by ReLU and fine-tune [4]. Compiling the graphs with these toolchains and running them on the devices is future work.

The strict graph needs only squares, the sigmoid and the inverse square root, so it runs whole on seven accelerators: Ethos-U55/U65, Ethos-U85, VeriSilicon VIP, the TI TDA4VM, the Rockchip RK3588, the Edge TPU and, in INT8, the Jetson DLA. On the Jetson DLA the inverse square root runs as an INT8 power and the reciprocal as an INT8 division, each with a constant operand, and softmax runs on the NPU on Orin. The R-Car CNN-IP lacks the inverse square root and runs it on the on-chip DSP. The open-source NVDLA IP and Ethos-N78 lack the inverse square root too and leave it, with the LayerNorm and the reciprocal that need it, to the host. The extended graph runs whole on Ethos-U, VeriSilicon, the TDA4VM and the RK3588, and on R-Car with the DSP. The toolchains of the other four have no matrix product of two activations and place the MatMul on the host, on three of them also softmax, and on two also the inverse square root of LayerNorm. The Edge TPU compiler adds a limit of its own: it splits a model only once, so the first unsupported operation and everything after it run on the CPU [11]. The strict graph runs whole there, and the extended graph moves to the CPU from the first attention score on. On NVIDIA these limits come from the toolchains, not the hardware: the open-source NVDLA IP, which the Jetson DLA is based on, has a programmable lookup table with linear interpolation, two tables of 65 and 257 entries, in its activation and normalization units [14]. It can hold every function of Φ_{strict} and Φ_{ext}, but both NVIDIA toolchains fill it only for the functions in the table.

Every accelerator multiplies two activations elementwise, so none is product-free in the sense of Definition 1: a CNN-class NPU provides the Hadamard basis $\mathcal{B}_\odot$. There, $QK^\top$ and PV need no pointwise work, since a broadcast Hadamard product followed by a fixed sum computes them (Proposition 2). With the sigmoid and the inverse square root, the Hadamard basis also runs the rest of the block (Proposition 1). The Hadamard graph takes the operations of the strict graph, with the elementwise product between two activations where the strict graph multiplies an activation by itself, so it runs on the accelerators where the strict graph runs. Where the toolchain multiplies only tensors of equal shape, as on the Ethos-N78, broadcast routing expands both operands first, as the strict graph does for its polarization operands. The Ethos-U55 compiler takes this route: it turns a matrix multiplication between activations into an

Supplementary Table S1. Documented NPU support of the dense compiled graphs. Whole-graph compilation on the devices is future work.

Accelerator	Unary maps on the NPU	Strict graph	Hadamard graph	Extended graph
Arm Ethos-U55/U65 [5]	Exp, Log, Rsqrt, GELU, Sigmoid, Tanh	Whole	Whole	Whole
Arm Ethos-U85 [5]	Exp, Log, Rsqrt, GELU, Sigmoid, Tanh	Whole	Whole	Whole
VeriSilicon VIP [6]	Exp, Log, Rsqrt, GELU, Sigmoid, Tanh	Whole	Whole	Whole
TI TDA4VM, C7x with MMA [7]	Exp, Log, Sqrt, GELU, Sigmoid, Tanh; power by a constant	Whole	Whole	Whole
Rockchip RK3588 [8]	GELU, Sigmoid, Tanh; power with exponent ± 0.5, 2 or 3	Whole	Whole	Whole
Renesas R-Car V4H/V4M, CNN-IP and DSP [9]	CNN-IP: ReLU, Sigmoid, Tanh	DSP: Rsqrt	DSP: Rsqrt	DSP: MatMul, softmax, LayerNorm, GELU
NVIDIA Jetson Xavier/Orin DLA [10]	ReLU, Sigmoid, Tanh, Abs, Sin, Cos, Atan; INT8 power and division by a constant	Whole	Whole	Host: MatMul
Google Edge TPU [11]	ReLU, Sigmoid, Tanh, Rsqrt	Whole	Whole	Host: MatMul, row-wise softmax
Open-source NVDLA IP [12]	ReLU, Sigmoid, Tanh	Host: Rsqrt	Host: Rsqrt	Host: MatMul, softmax, LayerNorm Rsqrt
Arm Ethos-N78 [13]	ReLU, Sigmoid, Tanh	Host: Rsqrt	Host: Rsqrt	Host: MatMul, softmax, LayerNorm Rsqrt

elementwise product into an int32 buffer followed by a sum, one output column at a time. The Ethos-U85 compiler runs the same operation on its MAC array. TopK runs on the TDA4VM accelerator and on the R-Car DSP. On the other accelerators top-k attention and the routed support keep the selection on the host; Ethos-U and VeriSilicon can instead replace TopK by k arg-max passes with masking.

Supplementary Note: Why the bound counts pointwise work

Write $\mathrm{IO}_L(G)$ for the minimum number of words moved between a fast memory of L words and slow memory, over all schedules of a graph G. With bandwidth β, every schedule takes time

$$T(G) \;\geq\; \max\Big(\mathrm{IO}_L(G)/\beta,\; \mathrm{MAC}(G)/R_{\mathrm{mac}},\; \mathrm{PW}(G)/R_{\mathrm{pw}}\Big).$$

The unfused strict graph writes the $N^2 d$ values $q_{if} + k_{jf}$ to memory (Supplementary Note, "Tensor sizes"), so this one schedule has arithmetic intensity $\Theta(1)$ and moves far more than IO_L. These writes are a property of the schedule, not of the basis. The strict expansion of Score in Lemma 5 has the access pattern of the matrix product it replaces, so the tiling of the matrix product applies. With square output tiles of side $\Theta(\sqrt{L})$, streaming over the reduction index,

and L small compared with the operands, a fused schedule moves

$$\mathrm{IO}_L = O\Big(\frac{N^2 d}{\sqrt{L}} + N^2 + Nd\Big)$$

words per head, where the last two terms are the compulsory traffic of writing the scores and reading Q and K; for classical (non-Strassen) schedules of the native product the first term is also a lower bound in this regime [15]. A fused strict kernel therefore has the same traffic order as native $QK^\top$. Fusion changes neither the number of unary evaluations nor the unit that must perform them. That is why Theorem 2 bounds PW and Corollary 1 rests on the third term. The same tiling applies to a Hadamard product followed by a fixed reduction (Proposition 2): fused, it has the access pattern of the matrix product, so the Hadamard and extended bases differ only in the schedule a compiler chooses.

Supplementary Note: Where the compiled graph spends its time

Supplementary Table S2 runs each graph eagerly, times the top-level submodules of every block with CUDA events and records their working memory, for DeiT-Tiny on Pet at $k = 16$ (bf16, 20 test batches of 32, RTX 4070 Ti SUPER). The working memory of a module is the peak memory it allocates above what is in use when it starts; the table reports the largest value over its calls. The strict graph adds 64.8 ms per batch over the extended graph; 73.3% of it falls in Score and Mix, and 9.8 ms in softmax. The module totals are 117.0 and 52.2 ms per batch, a ratio of 2.2. The profiled reference runs its attention unfused, with $QK^\top$, softmax and PV as separate kernels: it spends 9.0 ms there, against 4.1 ms and 2 MiB in its fused attention kernel, with a module total of 25.9 ms. At $k = 32$ and $k = 64$ the shares are 74.8% and 76.5%. Score costs the same at every k because scores are computed densely; Mix grows with k. The routed graph of Table 2 forms only the selected scores: Score falls from 50 to 11 ms per batch and its working memory from 469 to 77 MiB, and the centroid selection adds 9.6 ms.

The Hadamard graph of Proposition 1 forms the same broadcast grid as the strict graph, with one product per term instead of a sum and a square. Its Score takes 31.6 ms per batch, against 49.7 ms for the strict graph, and 471 MiB, the bf16 tensor of the $N^2 d$ products per head of one batch; the strict graph holds one tensor of the same size, since it squares its sums in place. Its module total is 78.2 ms. With routed support its Score falls to 9.2 ms and 78 MiB, the centroid selection adds 9.7 ms, and the module total is 65.2 ms.

The profile runs one kernel per node and synchronizes around every module, so it shows where eager time goes. Table 2 times the compiled forward pass on batches already on the GPU (Methods), where `torch.compile` fuses the elementwise chains of every graph. At $k = 16$ the compiled reference takes 2.20 ms per batch, and the extended, Hadamard and strict graphs take 7.21, 9.18 and 11.08 ms. The strict graph thus takes 1.54 times the time of the extended graph, against 2.2 in the eager module totals. Fused kernels keep the intermediates of a chain in fp32; all accuracies come from the eager bf16 runs.

Supplementary Note: Polarization and squares

The strict graph forms each product of two activations from squares, by one of two classical identities:

$$ab = \tfrac{1}{4}\big[(a+b)^2 - (a-b)^2\big], \qquad ab = \tfrac{1}{2}\big[(a+b)^2 - a^2 - b^2\big].$$

The first, polarization, is the method of quarter-squares [16]. The second, the norm expansion, forms products in expressivity proofs for ReLU networks [17].

Supplementary Table S2. Per-module GPU time and working memory at $k = 16$ in eager execution, summed over the 12 blocks of one batch. Routed: 32 query centroids. Ext.: the extended graph, with GELU; the other graphs use $x \odot \sigma(1.702x)$. Ref.: the reference with unfused attention. Softmax: top-k with the tail term, dense for Ref.

	Time (ms/batch)						Working memory (MiB)					
	Strict		Hadamard				Strict		Hadamard			
Module	top-k	routed	top-k	routed	Ext.	Ref.	top-k	routed	top-k	routed	Ext.	Ref.
Score ($QK^\top$)	49.7	10.7	31.6	9.2	4.0	4.1	469	77	471	78	14	14
Mix (PV)	7.4	11.8	4.9	7.0	5.6	2.8	81	81	78	78	39	12
LayerNorm	23.0	22.2	17.9	17.9	17.9	5.5	16	17	19	19	19	9
Softmax	24.9	24.0	14.0	12.7	15.1	2.1	15	18	9	11	8	28
Centroid selection	–	9.6	–	9.7	–	–	–	4	–	3	–	–
Linear (qkv, proj)	3.3	3.8	4.2	3.6	4.3	5.0	7	7	9	9	9	9
MLP	8.7	9.4	5.5	5.1	5.3	6.3	83	83	28	28	18	18

Cost. Polarization pays two squares per product. The norm expansion pays one square per product plus the squares a^2 and b^2 of the factors, so an isolated product costs two squares by polarization and three by the norm expansion. In a contraction each factor enters many products, and its square is formed once. $QK^\top$ of two $N \times d$ matrices then costs $2N^2d$ squares by polarization and $N^2d + 2Nd$ by the norm expansion, and the ratio to the bound of Theorem 2 falls from $2d$ to $d + 2d/N$. This is Winograd's precomputation of the self-terms of each row [18], with squares in place of products. The strict graph uses the norm expansion where the squares of the factors are shared, in $QK^\top$, PV, LayerNorm and the exact tail term, and polarization elsewhere, in softmax and the GELU approximation.

Exactness. Both identities are exact in real arithmetic and, without overflow or requantization, in integer arithmetic (Methods).

Rounding. Both identities recover a small product as the difference of large squares. In the value mix the result $y = \sum_k p_k v_k$ is a weighted mean of the values, while the squares sum to about $\sum_k v_k^2$; on DeiT-Tiny this ratio is 65 at $k = 16$ and 543 with all keys. In bf16 the order of the operations then sets the error (Supplementary Table S3). Rounding each partial sum before the subtraction, $\frac{1}{2}\big[\sum_k (p_k + v_k)^2 - \sum_k p_k^2 - \sum_k v_k^2\big]$, gives 57% relative error with all keys. Subtracting the two close squares per term inside the node that sums over keys gives 14% for either identity: $\frac{1}{4}\sum_k \big[(p_k + v_k)^2 - (p_k - v_k)^2\big]$ by polarization and $\frac{1}{2}\big[\sum_k \big(v_k^2 - (v_k - p_k)^2\big) + \sum_k p_k^2\big]$ by the norm expansion. With this order the two identities differ in cost, not in precision. As Brent showed for Winograd's identity [19], scaling the operands helps further: the fixed affine maps $p \to \lambda p$ and $v \to v/\lambda$ with $\lambda^2 = \|v\|/\|p\|$ cut the error to 2.7%.

Scaling and PTQ. Brent's scaling and the PTQ scales are the same choice. PTQ gives each tensor a grid with step $s = \max|x|/(2^{b-1} - 1)$ for b bits. The node that forms $a \pm b$ adds two tensors, so in integers both operands sit on one grid, and with the step set by the larger operand the smaller keeps few levels: in the mix the weights $p \le 1$ share the grid of values several times larger. The scaling $a \to \lambda a$, $b \to b/\lambda$ leaves the product unchanged and gives both operands the same range, so one grid fits both. It also makes the squares as small as they can be, since $\lambda^2 a^2 + b^2/\lambda^2 \ge 2|ab|$ with equality at $\lambda^2 = |b|/|a|$, and the size of the squares sets the step of the node that subtracts them. PTQ calibration already measures the ranges, so $\lambda^2 = \max|b|/\max|a|$ follows from the same statistics, and λ folds into the fixed affine maps that

produce a and b and into the output maps, so it costs nothing at run time. A node that combines several inputs, as the norm expansion does, quantizes each input on its own grid and rescales it before the sum (Methods); this takes care of the subtraction, but the balance of the operands inside $a \pm b$ still sets the number of bits. The strict graph in this paper applies no λ, and its fixed affine nodes take 12-bit input codes. Brent's scaling can reduce this bit width.

Supplementary Table S3. bf16 error of the value mix on DeiT-Tiny/Pet, relative to float64, averaged over the 12 blocks of 16 test images. Partial sums and scaled: emulated kernels with bf16 rounding of every stored value; λ is fitted on 8 other images. The cancellation factor is $\sum_k v_k^2/|y|$.

			Norm expansion		
k	Cancellation	Polarization, per term	partial sums	per term	per term, scaled
16	65	4.3%	9.1%	5.0%	1.0%
197	543	14.1%	57.4%	14.4%	2.7%

Supplementary Note: Tail-mass ablation on DeiT-Tiny

Supplementary Table S4 compares the tail modes on Pet before training, and after Jeffreys distillation with trainable q. A fixed small tail mass ($q = 10^{-3}$) behaves like tail drop: the omitted mass must be estimated. Supplementary Table S5 compares the calibrated q, a constant per block averaged on the training split, with the exact per-query tail mass on the held-out Pet test split, which no calibration or selection reads. Both sample 4096 queries per block from each minibatch of 32 images. The two means agree within 0.012, and within 0.001 for $k \geq 16$, while single queries spread widely around them.

This spread leaves room between the two modes. Calibrated mode uses $q = \mathbb{E}\tau$ and exact mode $q = \tau$. Between them lies the conditional mean $\mathbb{E}[\tau \mid \phi]$ for any statistic ϕ of what top-k attention reads, such as the log-sum-exp of the kept scores, the k-th score and the number m of dropped keys. Its expected squared error is never larger than that of the constant, since $\mathrm{Var}\tau = \mathbb{E}\big(\mathbb{E}[\tau \mid \phi] - \tau\big)^2 + \mathrm{Var}\mathbb{E}[\tau \mid \phi]$. Predicting the dropped partition sum as $m\,g(\phi)$, instead of τ itself, gives $q = 0$ when no key is dropped. It therefore also covers causal attention, where the tail mass grows with the position of the query and one constant per block fits poorly: on Pythia-70M at $k = 16$ the exact tail mass on 4 training minibatches is 0.062 ± 0.115, between 0 and 0.70.

Supplementary Table S4. Tail modes on DeiT-Tiny/Pet, top-1 accuracy (%). The last column is the saved one of five Jeffreys-distilled runs with trainable q, the run with the highest training accuracy.

k	tail drop	fixed $q = 10^{-3}$	calibrated	exact	calibrated, distilled
1	2.53	2.78	15.73	8.69	41.89
2	3.16	2.86	42.95	21.31	64.57
4	2.83	3.11	67.10	58.60	78.09
8	5.70	5.51	79.50	81.77	84.71
16	39.44	39.33	86.24	88.80	88.58
32	81.71	82.07	89.23	90.84	90.19
64	89.10	88.91	90.35	91.36	90.76

Supplementary Table S5. Calibrated against exact tail mass on DeiT-Tiny/Pet. Calibrated: the per-block constant from the training split, averaged over blocks. Exact: per-query tail mass on the held-out Pet test split, mean ± s.d. within a block averaged over blocks (min, max).

k	calibrated mean q, training	exact τ, test
1	0.7825	0.7943 ± 0.1540 (0.0007, 0.9925)
2	0.7139	0.7229 ± 0.1703 (0.0004, 0.9852)
4	0.6294	0.6349 ± 0.1839 (0.0004, 0.9710)
8	0.5278	0.5300 ± 0.1885 (0.0001, 0.9438)
16	0.4116	0.4125 ± 0.1770 (0.0000, 0.8915)
32	0.2826	0.2831 ± 0.1475 (0, 0.7920)
64	0.1511	0.1513 ± 0.0980 (0, 0.6052)
96	0.0811	0.0814 ± 0.0607 (0, 0.4286)
128	0.0394	0.0398 ± 0.0330 (0, 0.2627)
197	0	0

Supplementary Note: Distillation loss

Supplementary Table S6 compares cross-entropy and Jeffreys distillation on Pet. For $2 \leq k \leq 32$, Jeffreys distillation recovers 45% to 46% of the accuracy gap between the compiled model and the reference, and cross-entropy 36% to 43%. Jeffreys gains most, 1.2 to 1.5 points, at $k = 2$ and 4, since it matches the full teacher distribution and not only the label.

Supplementary Table S6. Distillation on DeiT-Tiny/Pet, top-1 accuracy (%) of the saved run, the one with the highest training accuracy, and mean ± s.d. over five seeds.

k	compiled	CE saved	CE mean	Jeffreys saved	Jeffreys mean
1	15.73	41.24	40.94 ± 0.24	41.89	41.22 ± 0.60
2	42.95	64.49	63.82 ± 0.93	64.57	65.30 ± 0.47
4	67.10	77.02	77.03 ± 0.18	78.09	78.22 ± 0.18
8	79.50	84.33	84.40 ± 0.18	84.71	84.91 ± 0.15
16	86.24	88.17	88.26 ± 0.06	88.58	88.56 ± 0.19
32	89.23	89.92	90.00 ± 0.19	90.19	90.18 ± 0.08
64	90.35	90.84	90.45 ± 0.22	90.76	90.43 ± 0.18
96	90.76	90.90	90.83 ± 0.09	91.25	90.86 ± 0.22
128	91.11	91.11	91.11 ± 0.00	91.11	91.11 ± 0.00
197	91.36	91.36	91.36 ± 0.00	91.36	91.36 ± 0.00

Supplementary Note: bf16 against fp32

Supplementary Table S7 compares bf16 with fp32 for every network with all keys. The bf16 column evaluates the saved models: weights in bf16 and arithmetic under CUDA autocast, as in the main tables. The fp32 column uses the fp32 reference weights and IEEE fp32 arithmetic. Each compiled graph is rebuilt in fp32, so its fixed coefficients are fp32 too. With all keys, no calibrated quantity is active. The DeiT-Tiny and Pythia-70M results repeat exactly between runs. MambaIRv2-Light uses nondeterministic kernels, and two runs differ by up to 0.01 dB.

In fp32, every compiled graph with exact GELU reproduces its reference (Theorems 1 and 3, Proposition 1). On Pet, the extended, strict and Hadamard graphs match the accuracy and loss of DeiT-Tiny to four digits. The extended graph matches on ImageNet-1k and WikiText-2, and on Set5 within the run-to-run variation. With $x\,\sigma(1.702x)$, the strict and Hadamard graphs compute the same model and give the same fp32 accuracy. bf16 changes the accuracy of DeiT-Tiny by at most 0.46 points and the PSNR of MambaIRv2-Light by at most 0.04 dB.

Pythia-70M runs in fp32. In bf16 its validation NLL rises from 4.55 to 6.68 nats, for the reference and the compiled graph alike. Only bf16 weights with fp32 arithmetic give 4.56. The degradation cause is the logits. They share a large offset that softmax ignores: their median is 1061, while the top two differ by a median of 0.73. Near 1000, bf16 resolves steps of 8, so it loses the ranking of the tokens. The output projection causes 1.7 of the 2.1 nats and attention the other 0.4.

Supplementary Table S7. bf16 against fp32 with all keys. Top-1 accuracy (%) for DeiT-Tiny, PSNR (dB) on the Y channel of Set5 for MambaIRv2-Light, validation NLL (nats) for Pythia-70M. Sigmoid GELU: $x\,\sigma(1.702x)$.

Network	Model	bf16	fp32	bf16 − fp32
DeiT-Tiny, Pet	reference	91.36	91.33	+0.03
	extended	91.36	91.33	+0.03
	strict, GELU	91.06	91.33	−0.27
	Hadamard, GELU	91.28	91.33	−0.05
	strict, sigmoid GELU	90.73	90.27	+0.46
	Hadamard, sigmoid GELU	90.35	90.27	+0.08
DeiT-Tiny, ImageNet-1k	reference	72.02	72.02	+0.01
	extended	72.01	72.02	0.00
MambaIRv2-Light ×2, Set5	reference	38.234	38.272	−0.038
	extended	38.231	38.270	−0.039
MambaIRv2-Light ×4, Set5	reference	32.511	32.512	−0.002
	extended	32.502	32.507	−0.006
Pythia-70M, WikiText-2	reference	6.681	4.550	+2.131
	extended	6.680	4.550	+2.130

Supplementary Note: Quantization with exact GELU

Table 3 reports each graph in its deployment configuration: the strict graph with $x\,\sigma(1.702x)$ and 12-bit inputs of its fixed affine nodes, before distillation, and the extended graph with GELU and 8-bit codes, after Jeffreys distillation. Supplementary Table S8 compares the strict graph with exact GELU and with $x\,\sigma(1.702x)$ under the same PTQ, both before distillation. The quantized graph has 344 nodes with GELU and 380 with its approximation, 332 and 368 at $k = 197$. Exact GELU matches no NPU configuration and serves as a comparison. With either GELU, PTQ costs the strict graph at most 1.5 points, against 2.5 to 3.9 points for the extended graph.

Supplementary Table S8. PTQ of the strict graph with exact GELU and with $x\,\sigma(1.702x)$ on DeiT-Tiny/Pet, before distillation, with the PTQ of Table 3. bf16: top-1 accuracy (%) of the float model. Δ: change after PTQ with per-channel or per-tensor weight scales. The approximation columns repeat Table 3.

	GELU			$x\,\sigma(1.702x)$		
k	bf16	Δ, per-channel	Δ, per-tensor	bf16	Δ, per-channel	Δ, per-tensor
16	86.37	−1.44	−0.90	84.08	−1.47	−1.06
32	89.07	−0.76	−0.68	87.98	−0.68	−0.71
64	90.35	−0.49	+0.05	89.78	−0.05	−0.14
197	91.06	−0.08	+0.19	90.73	−0.46	−0.98

Supplementary Note: Tensor sizes

Native $QK^\top$ reads $2Nd$ values per head and writes N^2. The unfused strict graph also writes the $N^2 d$ sums $q_{if} + k_{jf}$, squares them in place, and reduces over f. For DeiT-Tiny that is 2.48×10^6 values per head per image against 3.88×10^4, a factor $d = 64$, which drives the memory column of Table 2. Top-k mixing reads k of the N value rows per query, a fraction k/N of the value traffic of dense mixing.

References

[1] Strassen, V. Gaussian elimination is not optimal. *Numerische Mathematik* **13**, 354–356 (1969).

[2] Alman, J. *et al.* More asymmetry yields faster matrix multiplication. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005–2039 (2025).

[3] Bürgisser, P., Clausen, M. & Shokrollahi, M. A. *Algebraic Complexity Theory* (Springer, 1997).

[4] Mirzadeh, I. *et al.* ReLU strikes back: Exploiting activation sparsity in large language models. In *International Conference on Learning Representations* (2024). URL `https://arxiv.org/abs/2310.04564`.

[5] Arm. Ethos-U Vela compiler source code (regor backend), commit dd3f874. `https://gitlab.arm.com/artificial-intelligence/ethos-u/ethos-u-vela` (2026). Files `ethosu/regor/tflite/tflite_supported_operators_u55.cpp`, `_u85.cpp`, `architecture/ethosu55/ethos_u55_register_cs_generator.cpp`, `architecture/ethosu85/ethos_u85.cpp`. Accessed 2026-10-03.

[6] VeriSilicon. TensorFlow Lite VX delegate for VeriSilicon NPUs, commit b76047d. `https://github.com/VeriSilicon/tflite-vx-delegate` (2026). Operator map `op_map.cc` and `op_status.md`. Accessed 2026-10-03.

[7] Texas Instruments. edgeai-tidl-tools: Supported operators, commit 7cdef46. `https://github.com/TexasInstruments/edgeai-tidl-tools/blob/master/docs/operators.md` (2026). Accessed 2026-10-09.

[8] Rockchip. RKNN compiler support operator list, v2.3.2. `https://github.com/airockchip/rknn-toolkit2/tree/master/doc` (2025). Accessed 2026-10-09.

[9] Renesas. R-car hybrid compiler: Supported operators, commit e73e7d9. `https://github.com/renesas-rcar/renesas-rcar-HybridCompiler` (2025). Accessed 2026-10-09.

[10] NVIDIA. TensorRT documentation: DLA supported layers and restrictions. `https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/dla-layer-restrictions.html` (2026). Accessed 2026-10-03.

[11] Google Coral. TensorFlow models on the edge TPU. `https://www.coral.ai/docs/edgetpu/models-intro/` (2026). Accessed 2026-05-05.

[12] NVIDIA. NVDLA software stack, commit 79538ba. `https://github.com/nvdla/sw` (2019). Compiler layer types (`umd/core/include/nvdla/IType.h`) and CPU softmax (`umd/core/src/compiler/engine-ast/SoftMaxOp.cpp`). Accessed 2026-10-03.

[13] Arm. Arm Ethos-N driver stack, support library, commit 6834c82. `https://github.com/ARM-software/ethos-n-driver-stack` (2025). `driver/support_library/src/SupportQueries.cpp` and `SUPPORTED.md`. Accessed 2026-10-03.

[14] NVIDIA. NVDLA hardware architectural specification: LUT programming. `https://nvdla.org/hw/v1/ias/lut-programming.html` (2018). Accessed 2026-10-09.

[15] Hong, J.-W. & Kung, H. T. I/O complexity: The red-blue pebble game. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC)*, 326–333 (1981).

[16] Laundy, S. L. *Table of Quarter-Squares of All Integer Numbers, up to 100,000, by which the Product of Two Factors May Be Found by the Aid of Addition and Subtraction Alone* (Charles and Edwin Layton, London, 1856).

[17] Yarotsky, D. Error bounds for approximations with deep ReLU networks. *Neural Networks* **94**, 103–114 (2017).

[18] Winograd, S. A new algorithm for inner product. *IEEE Transactions on Computers* **C-17**, 693–694 (1968).

[19] Brent, R. P. Error analysis of algorithms for matrix multiplication and triangular decomposition using Winograd's identity. *Numerische Mathematik* **16**, 145–156 (1970).