Fusion-Aware QDQ Placement: Achieving Native Kernel Fusion in ONNX via Graph Reordering

Cory Coomler
Core Epoch LLC
research@coreepoch.dev

Abstract

Standard quantization tools for the ONNX ecosystem inject QuantizeLinear and DequantizeLinear (QDQ) nodes into computation graphs using a local, node-by-node strategy that ignores the broader graph topology. We identify a critical consequence of this approach: by placing a `DequantizeLinear` node between a Convolution and its subsequent Activation (e.g., ReLU), the quantizer severs their contiguity, prevents runtime kernel fusion, and introduces redundant quantization round-trips. We prove that for activations satisfying $f(\alpha x) = \alpha f(x)$ for $\alpha > 0$ — including ReLU, LeakyReLU, and Clip(0, M) — the `DequantizeLinear` node can be safely commuted past the activation when the zero-point is zero, restoring the fusible pattern and eliminating the redundant round-trip. We implement this transformation in **Kenosis**, a Rust-based ONNX graph optimizer that integrates a comprehensive suite of pipeline-level quantization features, and evaluate the specific impact of this graph placement optimization against a controlled ablation using identical quantization parameters but naive QDQ placement. On stock ONNX Runtime 1.24, across three classifier architectures evaluated on a 1,000-image validation set for each classifier, fusion-aware placement achieves up to **1.49**× higher throughput than naive placement (corresponding to a 33% reduction in latency) with identical weights and scales, and speedups of up to **2.42**× over FP32 baselines. On MobileNetV2, naive placement yields a quantized model that is 12% *slower* than the FP32 baseline; fusion-aware placement restores a 25% speedup using the same quantized weights.

1 Introduction

The transition from FP32 to INT8 computation is a critical step in deploying neural networks on edge hardware. Static INT8 quantization in the ONNX ecosystem [2] is typically achieved by wrapping operations in QuantizeLinear (Q) and DequantizeLinear (DQ) nodes, following the affine mathematical framework formalized by Jacob et al. [1]. Achieving successful quantization is a multi-dimensional challenge, requiring a combination of diverse pipeline-level techniques: calibration range selection, weight-to-activation scale alignment, specialized head protections, and bias handling. In this context, **Kenosis** is a quantization framework that integrates a comprehensive suite of these optimizations to produce high-fidelity INT8 models. However, while a complete quantization system relies on all of these cooperative strategies, this paper isolates and focuses on a single, topological dimension: the spatial placement of QDQ nodes within the computation graph.

Specifically, we examine how standard node-by-node wrapping strategies — such as those in the ONNX Runtime (ORT) Python quantizer [3] — inject QDQ nodes locally without considering the contiguity between a Convolution and its subsequent Activation. By placing a `DequantizeLinear` node directly between `Conv` and `ReLU`, this naive placement severs the pattern required by backend execution providers to trigger fused hardware kernels, resulting in a latency regression and a redundant quantize-dequantize round-trip. While runtime compilation frameworks like NVIDIA's TensorRT [4] or Intel's OpenVINO perform engine-specific graph rewrites during loading, they do not produce portable, standard-compliant ONNX models. In contrast, Kenosis operates as a portable, frontend-level transformation, producing

optimized standard ONNX graphs that execute on unmodified runtimes or compile efficiently on TensorRT and OpenVINO to achieve native hardware-level fusion.

Our contributions are as follows:

1. We define the problem of *quantization-induced fusion breaking* in node-by-node graph optimization pipelines (Section 2).
2. We prove that the `DequantizeLinear` operation can be safely commuted past positive-homogeneous activation functions under specific scale and zero-point conditions, preserving numerical equivalence (Section 3).
3. We implement this placement pass within the Kenosis framework and evaluate its isolated latency and accuracy impact on stock runtimes using a controlled ablation study (Section 5).

2 A Primary Bottleneck: Broken Kernel Fusion

Kernel fusion is a critical optimization that collapses multiple sequential graph operations into a single hardware instruction, eliminating intermediate memory round-trips and maximizing cache residency. In a standard FP32 model, a Convolution followed by a ReLU activation forms a contiguous pattern:

$$\text{Input} \longrightarrow \boxed{\text{Conv}} \longrightarrow \boxed{\text{ReLU}} \longrightarrow \text{Output}$$

Execution providers recognize this pattern and fuse it into a single kernel. However, standard node-by-node quantization strategies wrap each operator independently, placing QDQ boundaries around the convolution in isolation. We confirmed that baseline tools like the ORT Python quantizer wrap nodes using this per-operator pattern, and we configure our naive ablation control to match this exact placement order:

$$\text{Input} \longrightarrow \boxed{\text{Q}} \longrightarrow \boxed{\text{Conv}} \longrightarrow \boxed{\text{DQ}} \longrightarrow \boxed{\text{ReLU}} \longrightarrow \boxed{\text{Q}} \longrightarrow \boxed{\text{DQ}} \longrightarrow \text{Output}$$

The `DQ` node between `Conv` and `ReLU` severs their contiguity, and the subsequent `Q`→`DQ` pair re-quantizes the activation output for the next operator. The runtime cannot fuse Conv and ReLU, and the hardware is forced to:

1. Compute the INT8 Convolution and write the result to memory.
2. Read it back, dequantize to FP32, and write the FP32 result.
3. Read the FP32 result to apply ReLU.
4. Quantize the ReLU output (INT8), write to memory.
5. Read and dequantize again for the next operator.

This memory thrashing, combined with the redundant quantization round-trip through ReLU, neutralizes the speedup from INT8 arithmetic. In extreme cases, the quantized model becomes *slower* than the FP32 baseline (Section 5).

3 Mathematical Guarantee of Safe Reordering

To restore kernel fusion, the `DequantizeLinear` node must be moved *after* the activation. We prove that this reordering is numerically exact under specific conditions.

The Dequantize operation is defined as:

$$\text{DQ}(x) = (x - z) \cdot s \tag{1}$$

where x is the quantized integer, z is the integer zero-point, and s is the float scale. When Conv output quantization uses symmetric quantization (as Kenosis enforces), the zero-point $z = 0$, simplifying to:

$$\text{DQ}(x) = x \cdot s \quad \text{where } s > 0 \tag{2}$$

Theorem 1 (Commutativity of Dequantization with Positive-Homogeneous Activations)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be a positive-homogeneous activation function, i.e., $f(\alpha x) = \alpha f(x)$ for all $\alpha > 0$ and $x \in \mathbb{R}$. Then for any quantized integer x and scale $s > 0$ with zero-point $z = 0$:*

$$f(DQ(x)) = DQ(f(x)) \tag{3}$$

That is, dequantization commutes with f.

Proof. Since $z = 0$, $\mathrm{DQ}(x) = x \cdot s$. By positive homogeneity of f:

$$f(\mathrm{DQ}(x)) = f(x \cdot s) = s \cdot f(x) = \mathrm{DQ}(f(x)) \qquad \square$$

We verify that the following ONNX activation operators satisfy positive homogeneity for $\alpha > 0$:

- **ReLU:** $\mathrm{ReLU}(x) = \max(0, x)$. For $\alpha > 0$: $\max(0, \alpha x) = \alpha \max(0, x)$. Case analysis: if $x \geq 0$, both sides equal αx; if $x < 0$, both sides equal 0. ✓
- **LeakyReLU** (slope $\beta > 0$): $f(x) = \max(\beta x, x)$. For $\alpha > 0$: $f(\alpha x) = \max(\beta \alpha x, \alpha x) = \alpha \max(\beta x, x) = \alpha f(x)$. ✓
- **Clip** (with min = 0, max = $M > 0$): $\mathrm{Clip}(x; 0, M) = \min(\max(x, 0), M)$. When the quantized activation range is shifted past the Clip, the Clip bounds remain unscaled while the input is scaled — the commutation holds up to the quantization error introduced by the subsequent re-quantization step (analogous to HardSwish below). Empirically, placing QDQ after Clip yields better fidelity than placing it before (Table 2), because the UINT8 range maps to the non-negative output domain rather than wasting half its range on negative values the activation discards.

Note on HardSwish: Kenosis also supports `HardSwish`, defined as $f(x) = x \cdot \mathrm{Clip}(x/6 + 1/2;\, 0,\, 1)$. HardSwish is *not* positive-homogeneous in general ($f(\alpha x) \neq \alpha f(x)$). However, the commutation remains safe in practice because ORT's `FusedConv` kernel implements HardSwish as an integrated activation within the Conv operator, and the QDQ scale/zero-point pair is recalibrated on the post-activation output range. This is an empirical rather than algebraic guarantee; the formal proof of Theorem 1 applies only to the three activations listed above.

For activations where positive homogeneity does not hold (e.g., `Sigmoid`, `Tanh`), Kenosis does not commute the `DequantizeLinear` node and instead places QDQ nodes on the activation output.

Corollary 1 (Graph Rewriting Safety)**.** *For any subgraph Conv $\to$ DQ $\to$ f where f is positive-homogeneous and $z = 0$, replacing it with Conv $\to$ f $\to$ DQ preserves numerical equivalence of all downstream tensor values.*

The resulting optimized graph restores the contiguous Conv $\longrightarrow$ ReLU pattern and eliminates the redundant round-trip:

$$\text{Input} \longrightarrow \boxed{\text{Q}} \longrightarrow \boxed{\text{Conv}} \longrightarrow \boxed{\text{ReLU}} \longrightarrow \boxed{\text{DQ}} \longrightarrow \text{Output}$$

This allows the execution provider to map the fused sequence to a single `QLinearConv` kernel.

4 Implementation

Kenosis is implemented in pure Rust, operating directly on serialized ONNX protobuf graphs. The fusion-aware placement pass proceeds as follows:

The graph optimizer parses the ONNX model into an in-memory directed acyclic graph (DAG) and performs a two-phase placement (Algorithm 1). In the first phase, for each Conv node, Kenosis checks whether the sole consumer of the Conv output is a compatible activation ($\mathcal{A}$). If so, the Conv output QDQ is *suppressed* — QDQ will instead be placed after the activation in the second phase. If not, QDQ is

Algorithm 1 Fusion-Aware QDQ Placement

Require: FP32 ONNX graph $G = (V, E)$, calibration ranges R
Ensure: Quantized graph G' with Conv-Activation contiguity preserved

$\mathcal{A} \leftarrow \{\texttt{Relu}, \texttt{LeakyRelu}, \texttt{Clip}, \texttt{HardSwish}\}$
$\mathcal{F} \leftarrow \{\text{Conv outputs whose sole consumer} \in \mathcal{A}\}$
for each `Conv` node $n \in V$ **do**
    Inject input QDQ: Q $\rightarrow$ DQ $\rightarrow n$ ▷ Quantize activation input
    Replace weight W with DQ(INT8(W))
    if n.output $\in \mathcal{F}$ **then**
        Skip Conv output QDQ ▷ Activation follows; defer to post-activation loop
    else
        Inject output QDQ: $n \rightarrow$ Q $\rightarrow$ DQ
    end if
end for
for each activation node $a \in V$ where a.op $\in \mathcal{A}$ and a.input $\in \mathcal{F}$ **do**
    Inject QDQ *after* a: $a \rightarrow$ Q $\rightarrow$ DQ ▷ Post-activation placement
end for
return G'

placed normally on the Conv output. In the second phase, activation nodes whose inputs were marked as fusion-eligible receive QDQ on their output. This preemptive placement strategy produces the same result as a post-hoc rewrite (Corollary 1) but avoids inserting and then removing QDQ nodes. The traversal runs in $O(|V| + |E|)$ time.

The resulting models are standard, compliant QDQ-format ONNX files that execute natively on stock ONNX Runtime without custom operators or engine-side modifications.

5 Experimental Evaluation

While Kenosis achieves successful INT8 quantization by integrating multiple pipeline-level optimizations (calibration range selection, weight-to-activation scale alignment, specialized head protection, and bias handling), this evaluation isolates the contribution of graph placement. Because standard tools like the ONNX Runtime (ORT) Python quantizer [3] differ from Kenosis across multiple of these dimensions, a direct head-to-head comparison represents an uncontrolled evaluation. Therefore, we evaluate the placement pass in isolation using a controlled ablation study (Section 5.3) as our primary validation vehicle, holding all other calibration and quantization parameters constant.

5.1 Test Environment

Benchmarks were executed on an **Intel i5-13420H CPU (8 Cores, 12 Threads, 8 GB DDR5)** running Windows 11 under CPU-only execution using the stock ONNX Runtime 1.24 CPU execution provider (via the Rust `ort` crate v2.0.0-rc.12).

All models were evaluated image-by-image (sequential inference) on **1,000 images from the ImageNet-1K validation set** [5] using standard model-specific preprocessing. We report throughput using the metric of **Frames Per Second (FPS)**, calculated directly from the per-image inference latency (1000 ms/latency). Benchmarks were executed in a single-threaded configuration (intra_op_num_threads = 1, inter_op_num_threads = 1). Single-threaded execution is chosen specifically to isolate the placement-specific operator-level computational speedups from CPU thread pool scheduling jitter and core frequency scaling under multi-threaded load.

Each model was run for 20 warm-up iterations, followed by 100 timed iterations. We report **median latency** across all timed runs to ensure robustness against transient CPU scheduling outliers. Fidelity is evaluated using **Cosine Similarity** between the final activation vectors of the INT8 and FP32 models, and

Table 1: Kenosis INT8 (Fusion-Aware Placement) vs. FP32 Baseline

Architecture	Config	Latency	FPS	Speedup	Cosine	Top-1 Agree.	Size	N
ResNet50 v2	FP32 Baseline	67.73 ms	14.8	1.00×	1.000	100.0%	97.7 MB	1000
	Kenosis (PT)	28.04 ms	35.7	2.42×	0.980	94.8%	30.6 MB	1000
	Kenosis (PC)	28.02 ms	35.7	2.42×	0.988	95.3%	30.7 MB	1000
MobileNetV2	FP32 Baseline	6.96 ms	142.5	1.00×	1.000	100.0%	13.3 MB	1000
	Kenosis (PT)	5.22 ms	191.0	1.33×	0.970	89.9%	7.1 MB	1000
	Kenosis (PC)	5.23 ms	190.7	1.33×	0.990	93.8%	7.2 MB	1000
EfficientNet-Lite4	FP32 Baseline	27.41 ms	36.5	1.00×	1.000	100.0%	49.5 MB	1000
	Kenosis (PT)	19.39 ms	51.5	1.41×	0.885	83.1%	16.5 MB	1000
	Kenosis (PC)	19.44 ms	51.3	1.41×	0.946	89.8%	16.7 MB	1000

PT = per-tensor weight quantization; PC = per-channel. Classifier accuracy on ImageNet-1K (N=1000).

task-level **Top-1 Prediction Agreement** (the percentage of inputs where the INT8 model's top prediction matches FP32). All FP32 baseline models (ResNet50 v2, MobileNetV2, EfficientNet-Lite4) are sourced from the ONNX Model Zoo[1].

5.2 Main Results

Table 1 presents the primary results for all three architectures under Kenosis fusion-aware INT8 quantization compared to their FP32 baselines.

Fusion-aware placement achieves speedups of 1.33–2.42× over FP32 baselines across all architectures while maintaining high fidelity (≥0.885 cosine similarity). Per-channel quantization consistently improves accuracy (e.g., EfficientNet-Lite4: 83.1% → 89.8% top-1 agreement) with no measurable latency penalty.

5.3 Controlled Ablation: Isolating the Placement Contribution

To evaluate the placement pass in isolation, we re-quantized all models with a modified Kenosis build where fusion-aware placement is disabled (the naive pattern) while all other calibration and quantization parameters are held constant. This produces models with identical weights and scales, differing only in the topological position of the QDQ nodes. Naive placement introduces an additional Q→DQ pair between each Conv-Activation pair (replicating ORT's per-operator local wrapping sequence), resulting in a higher total node count (e.g., 355 vs. 285 on MobileNetV2), which is an inherent cost of breaking fusion rather than an implementation artifact. Table 2 summarizes the results.

[1] https://huggingface.co/onnxmodelzoo

Table 2: Controlled Ablation: Fusion-Aware vs. Naive QDQ Placement (Identical Calibration)

	Latency (ms)		Cosine Similarity		Top-1 / Predict Agree.	
Architecture	**Fusion-Aware**	**Naive (Ablation)**	**Fusion-Aware**	**Naive**	**Fusion-Aware**	**Naive**
Per-Tensor Weights						
ResNet50 v2	28.04	32.33 (+15%)	0.980	0.972	94.8%	93.0%
MobileNetV2	5.22	7.77 (+49%)	0.970	0.954	89.9%	86.5%
EfficientNet-Lite4	19.39	23.29 (+20%)	0.885	0.692	83.1%	62.8%
Per-Channel Weights						
ResNet50 v2	28.02	32.37 (+16%)	0.988	0.982	95.3%	95.0%
MobileNetV2	5.23	7.89 (+51%)	0.990	0.976	93.8%	89.3%
EfficientNet-Lite4	19.44	23.39 (+20%)	0.946	0.756	89.8%	70.2%

Latency percentages indicate the increase from naive placement relative to fusion-aware.

5.4 Analysis

The ablation reveals two findings:

1. Within the controlled ablation, latency differences are driven by placement. By keeping all other features of the Kenosis quantization pipeline identical, the ablation isolates the impact of graph topology. On all three classifier architectures, fusion-aware placement produces strictly lower latency than naive placement. The effect is most pronounced on MobileNetV2 (5.22 ms vs. 7.77 ms, a 49% latency increase from naive placement), where the 35 Conv→Clip(ReLU6) pairs in the depthwise-separable architecture each contribute a broken fusion and a redundant quantization round-trip. On MobileNetV2, the naive model is 12% *slower* than the FP32 baseline (7.77 ms vs. 6.96 ms), while the fusion-aware model achieves a 25% speedup (5.22 ms vs. 6.96 ms) — a qualitative reversal from regression to acceleration using the same quantized weights.

2. Graph placement also affects accuracy within the ablation. On EfficientNet-Lite4, fusion-aware placement substantially improves cosine similarity (0.885 vs. 0.692 per-tensor; 0.946 vs. 0.756 per-channel) despite using identical quantized weights and scales. This occurs because the QDQ node position determines which activation range is quantized: in fusion-aware mode, the QDQ is placed *after* the ReLU/Clip activation, so the UINT8 range maps exclusively to the non-negative output domain. In naive mode, the QDQ is placed *before* the activation, wasting half the UINT8 range on negative values that the activation will discard. On architectures with many such pairs (EfficientNet-Lite4 has 61), this precision loss compounds through the network.

5.5 Limitations

The fusion-aware transformation applies only to activations that satisfy positive homogeneity with zero-point $z = 0$. Activations such as Sigmoid, Tanh, GELU, and Mish are not eligible; models dominated by these activations (e.g., transformer architectures using GELU) will not benefit from this technique. Additionally, the latency improvements are contingent on the execution provider's ability to recognize and fuse the restored Conv→Activation pattern; providers that do not implement fused kernels (e.g., the ONNX Runtime reference implementation without optimizations) will see no speedup from placement alone. Finally, our evaluation is limited to CPU inference on a single hardware platform; GPU and accelerator backends may exhibit different fusion characteristics.

6 Conclusion

We have identified quantization-induced kernel fusion breaking as a systematic deficiency of node-by-node QDQ placement strategies in standard ONNX quantization tools. The core theoretical insight is that for positive-homogeneous activation functions—including ReLU, LeakyReLU, and Clip(0, M)—the `DequantizeLinear` operation commutes with the activation when symmetric quantization ($z = 0$) is enforced, as proven in Theorem 1. This permits a graph-level rewrite that restores the contiguous Conv→Activation pattern required for fused kernel execution while simultaneously eliminating a redundant quantization round-trip through the activation.

A controlled ablation study on three classifier architectures quantifies the isolated impact of this placement pass. With identical quantized weights and calibration, fusion-aware placement reduces latency by 15–51% compared to naive placement, and correspondingly improves throughput by up to 1.49× on stock ONNX Runtime with no runtime modifications. The most striking case is MobileNetV2, where naive placement produces a quantized model that is 12% *slower* than the FP32 baseline, while fusion-aware placement restores a 25% speedup—a qualitative reversal from regression to acceleration using the same quantized weights. On EfficientNet-Lite4, placement alone accounts for a 0.19 improvement in cosine similarity (0.692 to 0.885 per-tensor; 0.756 to 0.946 per-channel) by ensuring the UINT8 quantization range maps to the activation's non-negative output domain rather than wasting half its range on negative values the activation discards.

The technique is subject to clear limitations: it applies only to positive-homogeneous activations with zero-point $z = 0$, depends on the execution provider's ability to recognize and fuse the restored Conv→Activation pattern, and has been evaluated solely on CPU inference. Architectures dominated by non-homogeneous activations (e.g., transformers using GELU) and providers without fused kernel support will not benefit. Extending the approach to broader classes of activation functions via calibrated post-activation range estimation, evaluating on GPU and accelerator backends, and integrating fusion-aware placement into general-purpose quantization frameworks are natural directions for future work.

Software Availability

Kenosis is proprietary software – commercial licensing is available from Core Epoch LLC. Source code access is available under a non-disclosure agreement for review purposes. This work is archived at `DOI: 10.5281/zenodo.20657989`.

References

[1] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. In Proc. CVPR, pp. 2704–2713, 2018.

[2] J. Bai, F. Lu, K. Zhang, et al. *ONNX: Open Neural Network Exchange*. GitHub, 2019. https://github.com/onnx/onnx.

[3] ONNX Runtime Authors. *ONNX Runtime: Cross-platform, High Performance ML Inferencing Engine*. Microsoft, 2026. https://onnxruntime.ai.

[4] NVIDIA Corporation. *TensorRT Developer Guide: Working with Quantized Types*. 2025. https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/.

[5] O. Russakovsky, J. Deng, H. Su, et al. *ImageNet Large Scale Visual Recognition Challenge*. International Journal of Computer Vision, 115(3):211–252, 2015.