Optimized C++ Inference in SOFIE for Quantized ML Models

CERN openlab Summer Student Programme Lightning talks

Shaun Christopher Lee

Supervisors: Sanjiban Sengupta, Lorenzo Moneta

SOFIE

A Tool for Fast ML Inference within ROOT

- TVMA/ROOT tool that takes a trained neural network and generates standalone C++ inference code.
- Portability support targeting heterogeneous architectures: CUDA, HIP/ROCm, CPU, in the SOFIE standalone version hosted on the ML4EP/SOFIE github.
- ONNX model → Parser → Internal Model → Code Generation → C++ CPU / Alpaka CUDA

ONNX and Representing Quantization

Brief Introduction to Quantization

- Quantization stores each value as one of a small set of levels (a grid), with a scale fitted to that tensor. Faster on hardware, less traffic to move.

- At 8 bits, weights and activations are 4x smaller and cost 4x less memory traffic.
- Modern GPUs carry dedicated low-precision matmul units faster than general cores.
- Same mechanism covers 4-bit integer/float, etc.

$$\hat{x} = s \cdot Q(x/s)$$

	integer (INT8)	float (FP8 E4M3)
fitted to the data	scale s and offset z	scale s only
grid Q	$s(q - z)$ q: 8-bit integer, $-128 \ldots 127$ 256 levels, evenly spaced	$s(\pm 2^E(1 + m/2^p))$ $p = 3$ mantissa bits, so $m = 0 \ldots 7$; E: 4 bits 256 levels, spacing doubles each power of 2
scale s	sets the gap between levels	rescales the tensor into the format's range
rounding error	absolute: $\lvert x - \hat{x} \rvert \le s/2$	relative: $\lvert x - \hat{x} \rvert \le 2^{-(p+1)} \lvert x \rvert$ $= 6.25\%$

ONNX and Representing Quantization

Interchange Format for Trained ML Models

ONNX:

- A graph of standard operators (e.g. Gemm, MatMul, Conv, Relu, Softmax) and initializers (e.g. weights) that make up a trained model.
- Designed as a general representation for other tools to agree on. Export a model to ONNX format.

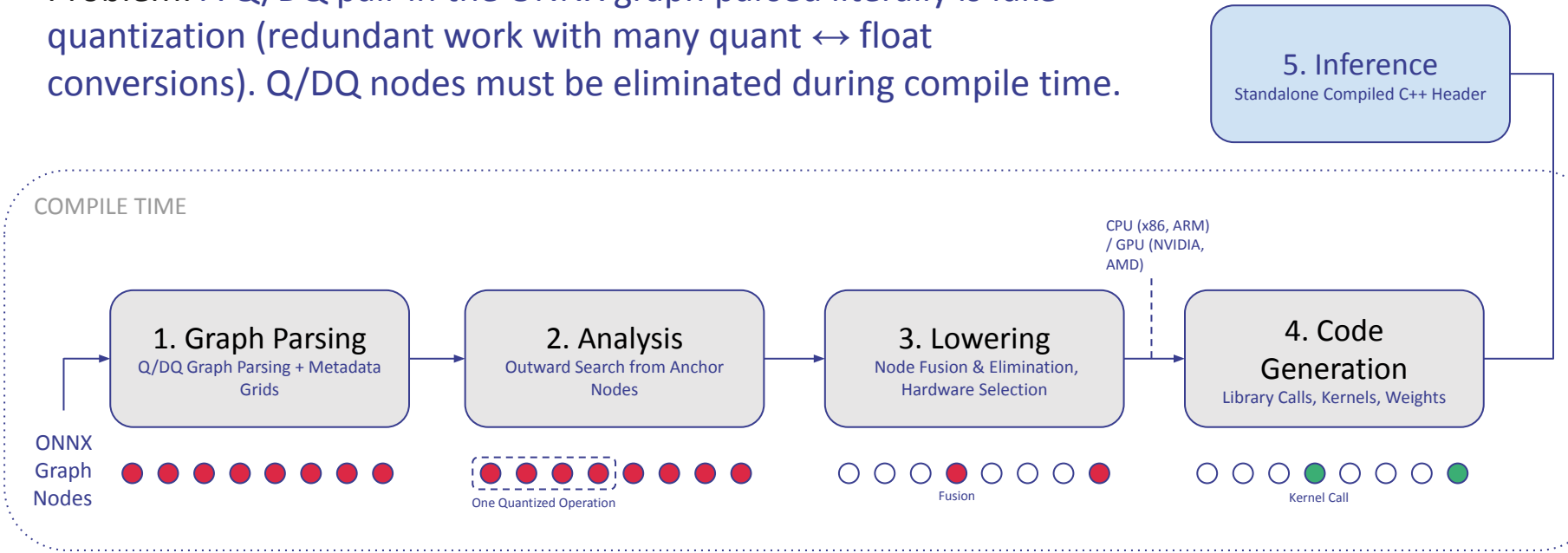
Q/DQ & QONNX:

- A quantized model reaches SOFIE as ONNX, annotated by what trained it. Two conventions: Q/DQ and QONNX.
- ONNX represents quantization as node pairs (Q/DQ). Tensor op nodes stay float, while Q/DQ pairs mark where the representation changes. QONNX represents the same information in Quant nodes.

SOFIE Quantization Pipeline

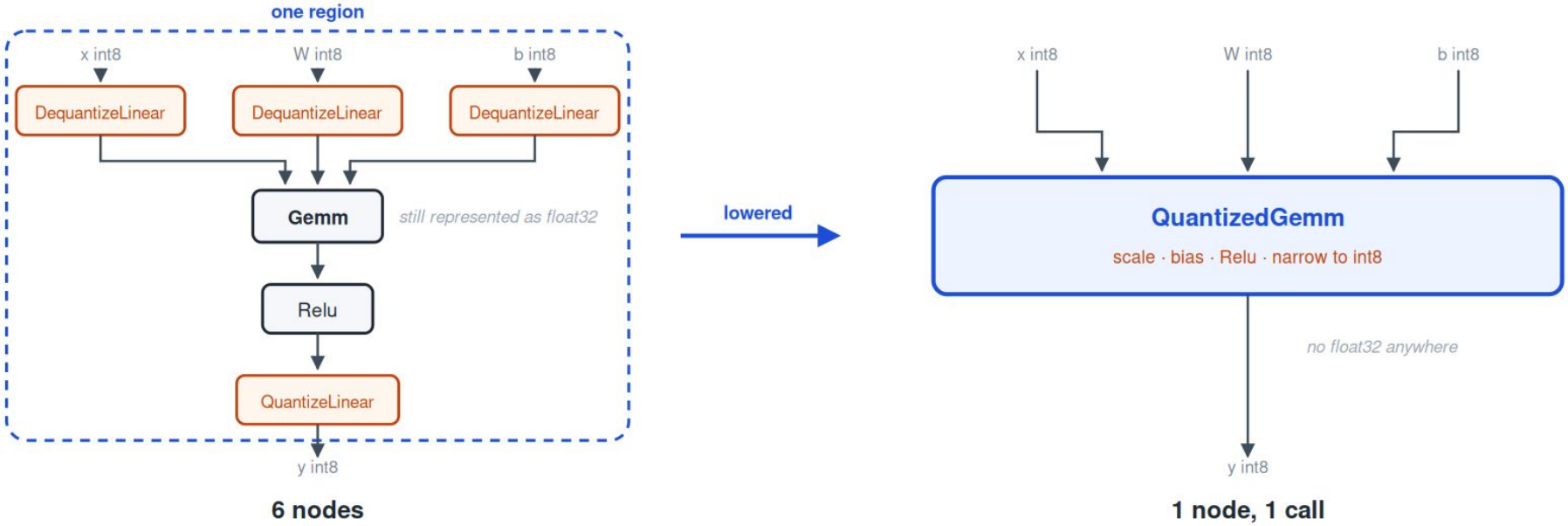
End-to-End Execution

Problem: A Q/DQ pair in the ONNX graph parsed literally is fake quantization (redundant work with many quant ↔ float conversions). Q/DQ nodes must be eliminated during compile time.



SOFIE Quantization Pipeline

Expanded Analysis + Lowering View



Experiments & Method

Measuring Latency & Speedup

- *20 warmup iterations (kernels), 250 ms budget. Timed with CUDA events on a single stream, median reported.*
- *FP32 baselines are native-export with no Q/DQ nodes. Compared against INT8/FP8 equivalent graphs with Q/DQ nodes specified.*
- *Experiment fixtures (operators or models) are structural and untrained (random initialized weights). No accuracy measured.*
- *Correctness measured within INT8/FP8 tolerance relative to FP32 (numerics differ slightly up to expected tolerance, large divergences discarded as incorrect implementation).*
- *Hardware: single H100, batch size 32 throughout.*

Operator Performance

GEMM (General Matrix Multiply), Conv, & Elementwise Add/Mul

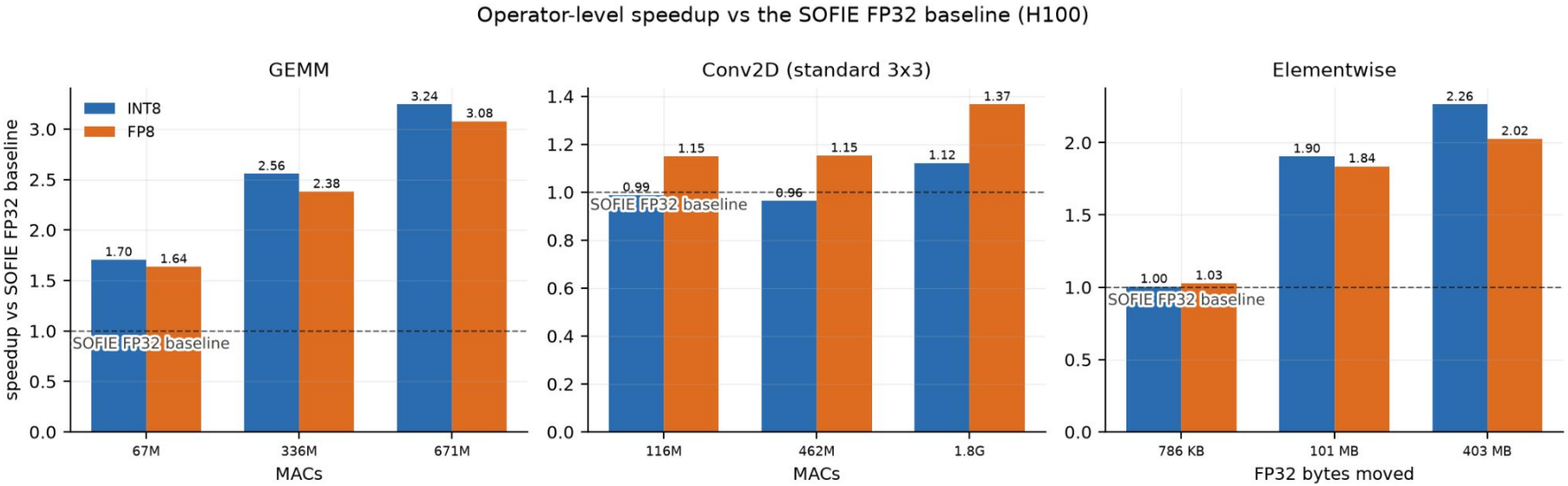


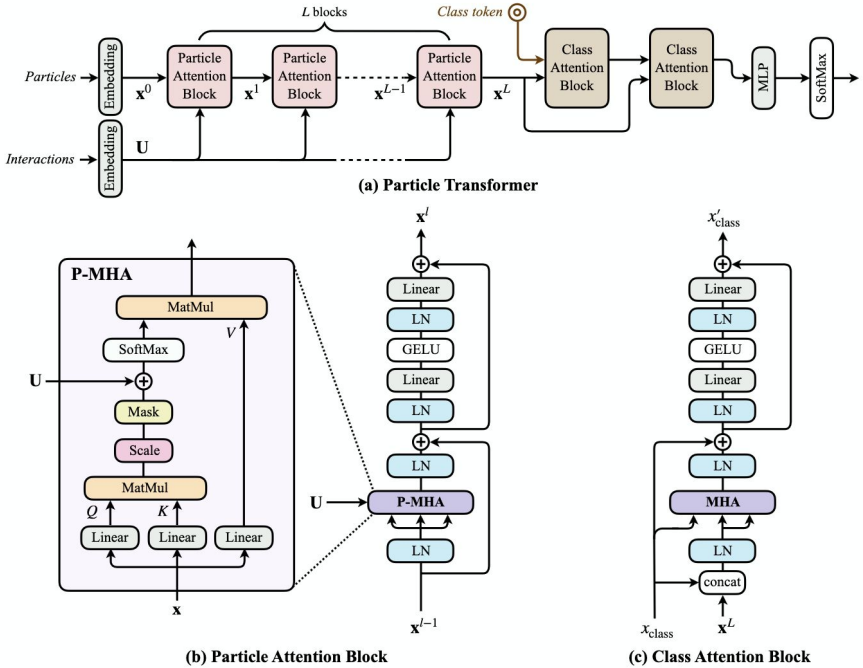
Figure 1. Operator-level speedup over FP32, H100. Each operator is measured by the amount of work performed / tensor problem size (fundamental units, compute-bound: MACs = multiply accumulates, memory-bound: bytes moved).

Experiments & Method

Particle Transformer (ParT) Architecture

The Particle Transformer. Qu, Li & Qian (2022)

- *A Transformer for jet tagging. Given the particles produced in a collision, identify which parton they came from.*
- *Particles represented as token embeddings. Tests operators like GEMM/MatMul, Softmax, LayerNorm, Relu/Gelu, end-to-end.*



End-to-End Performance

Particle Transformer (ParT)

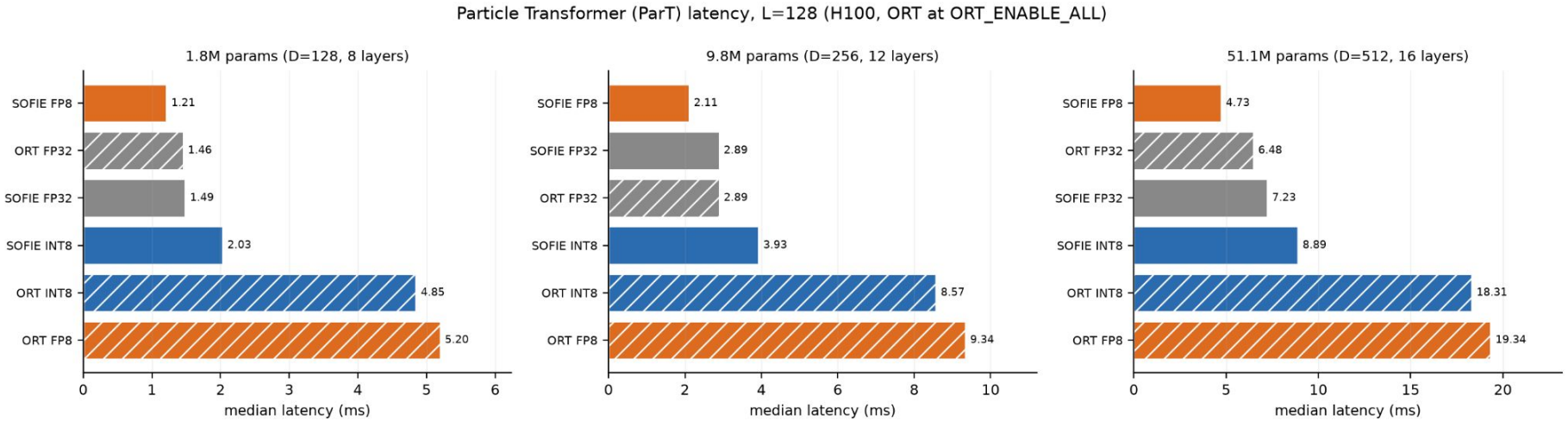


Figure 2. Median latency full-model ParT inference on one H100, L=128, at three model sizes, with FP32 as a native-export graph. Inputs are resident on the device, with transfers and session setup are excluded. ONNX Runtime (ORT) runs at full optimization level on FP32, INT8, FP8. ORT INT8/FP8 are the slowest throughout despite full optimization.

End-to-End Performance

Particle Transformer (ParT)

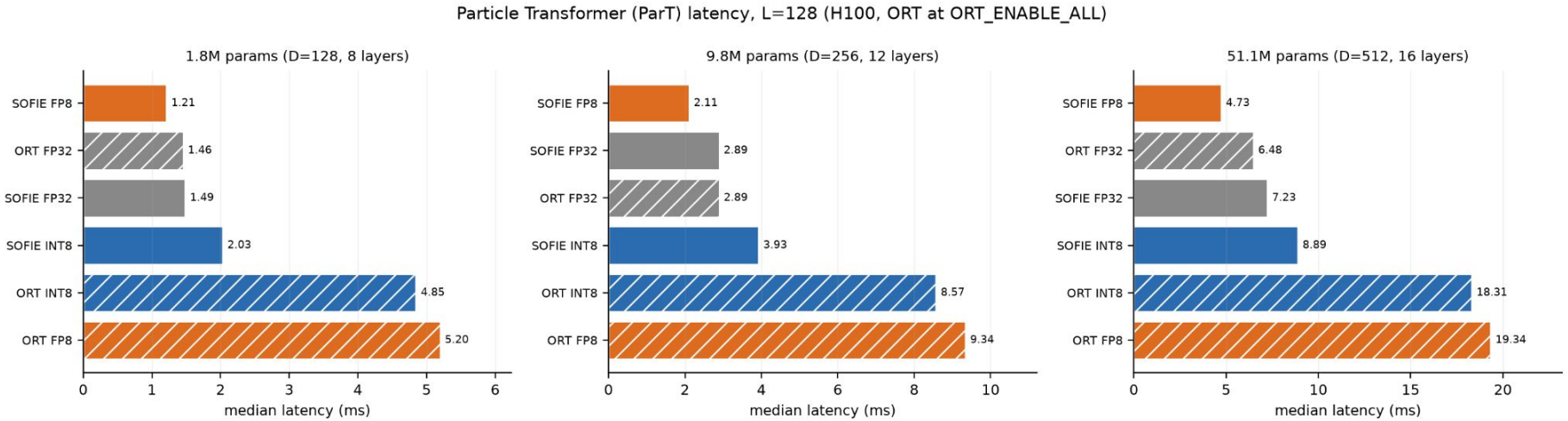


Figure 2. SOFIE and ORT run on TF32 tensor cores (H100). The optimization gap between INT8/FP8 and ORT FP32 comes from operators like Softmax, LayerNorm, Reshape, remaining Q/DQ nodes, suggested by GPU kernel time shown on the right.

	GEMM	non-GEMM
ORT FP32	32.9%	67.1%
SOFIE FP32	29.9%	70.1%
SOFIE INT8	18.7%	81.3%
SOFIE FP8	23.9%	76.0%

Summary

Optimized C++ Inference in SOFIE for Quantized ML Models

In-Place:

- *One first-class low-precision/quantization IR imports Q/DQ ONNX & QONNX, and represents int8 and fp8 as one grid abstraction.*
- *Quantized, fused operators lower to cuBLASLt / fused Alpaka CUDA.*
- *MLP & Transformer tested / supported end-to-end. Speedup improves as sequence length / model scales.*

Summary

Optimized C++ Inference in SOFIE for Quantized ML Models

Future Work:

- *The core FP32 path may differ in terms of fusion rules (e.g. separate bias, adds, layernorms) and tensor cores, so direct latency comparisons are not one-to-one. Both core and quantized paths have further optimization headroom.*
- *Only the NVIDIA/CUDA path for SOFIE_quantization is in place. Further work planned for CPU (x86, ARM) and AMD GPUs.*
- *Trained model evaluation end-to-end, with accuracy/latency reporting at each precision, and further dtype support.*
- *Operator coverage is Transformer/CNN-shaped. Further coverage planned for GNNs and State Space Models (Mamba-2).*

Thanks