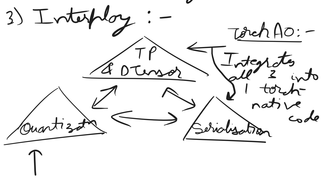
Gaps:- 1) Large batch size (memory)
2) Better speed (compute)

Problems:- 1) Perf drops w/t large batch size

2) Types of quantization are limited
↳ Eg:- Symmetric Quant only kernels

3) Interplay :-



GenLite provides new types of quantization
2) Perf for large batch size

#Experiment :- Llama 3.1 -8B w/t diff batch sizes & TP sizes.

Compute :- 8 XH100
for decode

Baseline → BF16
torch.compiled model

Batch size : 1, 32

TP size : 1, 4

Quant → 1) int4 weight only w/t group size 64
2) fp8 per row dynamic quant

(batch sz, tp)

(1,1) → 131 tks/s (1.27x) → 166 tks/s

(32,4) → 5575 tks/s (1.10x) → 6159 tks/s

(32,1) → 2799 tks/s → 3586 tks/s
(1.28x)

Supports both H100 & MI00s

* GenLite supports both packed & non-packed bit formats → fp8
(uint1, uint2 ...)
these are packed together for storage.

GenLite kernels Adapt across →
1) Small Batch size → GEMV for batch 1
→ Memory Bound
2) Large Batch Size → GEMM
→ Compute Bound
↔ 2-64 batch ⇒ GEMM-SPLIT K

Single Sample Inference :-

* GEMV kernels → GEMV-Split k for packed data

Batched → GEMM - Split k

* 32-64 → Compute bound

Optimal split hyperparam (k)
→ Range: 1-16
→ If k=1 ⇒ Normal GEMM kernel
↓
suitable for compute bound setting of 32 to 64 batch size

* Quantize first, shard later.

Follow-up :-

(1) GEMM-SPLIT K paper cause it forms basis of GEMLITE

(2) Binary & Ternary ANNs: efficient MatMul papers.