HW9

Tguo67

Question(s):

Read the related materials, and answer the following questions:

1. Why are W4A16, W4A4, and W8A8 not efficient?
2. How does QServe mitigate the cost of dequantization? You may answer from both QoQ quantization and QServe system.
3. How does COMET make one step forward in W4A4KV4 serving?

Related materials:

QServe paper: https://arxiv.org/pdf/2405.04532Links to an external site.

COMET paper: https://dl.acm.org/doi/pdf/10.1145/3676641.3716252Links to an external site.

Answers:

1. Why are W4A16, W4A4, and W8A8 not efficient?

- W4A16 (weight-only INT4, FP16 acts): Current GPU kernels must dequantize weights/partial-sums inside the GEMM main loop, pushing a lot of work onto slow CUDA cores instead of tensor cores and this dequant work dominates runtime
- W4A4 (both INT4): State-of-the-art W4A4 serving (e.g., Atom) also performs main-loop dequantization and extra register traffic, leading to 20–90% runtime overhead, again the loop is CUDA-core bound rather than tensor-core bound
- W8A8 (both INT8): While its main loop is tensor-core only (fast), it doesn't exploit the higher throughput of INT4 tensor cores and doesn't shrink KV cache, limiting end-to-end serving gains at scale compared with better co-designed schemes. (QServe picks W4A8KV4 as a superior throughput/memory point for serving

2. How does QServe mitigate the cost of dequantization? You may answer from both QoQ quantization and QServe system

QoQ quantization (algorithm side):

- Progressive group quantization (INT8→INT4): First quantize weights to INT8 with FP16 per-channel scales, then further to INT4 with per-group zero-points/scales. At runtime, execute W4A8 GEMMs on INT8 tensor cores, avoiding heavy main-loop dequant on CUDA cores
- Safe "Sub-after-Mul" dequant order + register-level parallelism (RLP): The progressive ranges guarantee no overflow, enabling vectorized register ops and turning dequant into cheap packed ops
- SmoothAttention + KV4: 4-bit KV cache with an attention fix (SmoothAttention) preserves accuracy so the memory savings become measured speedups in fused attention.

QServe system (kernel/runtime side):

- Compute-aware weight reordering: Store weights in the order consumed by the GEMM, minimizing pointer arithmetic and restoring wide (128-bit/thread) loads even with 4-bit storage.
- Fast dequant path (Mul→Sub) and INT8 tiling: Implements the progressive scheme so dequant happens as cheap vector ops while the main math stays on INT8 tensor cores.
- Attention made memory-bound again: With KV4 and kernel tweaks, fused attention shifts back to a memory-bound regime, turning KV compression into real tokens/s gains. (Ablations show much lower dequant overhead vs. Atom)

3. How does COMET make one step forward in W4A4KV4 serving?

COMET's step forward is to make W4A4 activations practical at scale (not just W4A8), while still shrinking KV to 4-bit.

- **FMPQ (Fine-grained Mixed-Precision for activations/KV):** Most activations become 4-bit, with outliers kept 8-bit; tiling + channel permutation align quant granularity with GPU compute units and keep accuracy.
- **COMET-W4Ax kernel:** Mixed-precision GEMM that runs W4A4 directly on tensor cores and converts INT4→INT8 efficiently for W4A8 tiles, packing data and using a software pipeline to hide conversion/load costs.
- **Mixed-precision data layout + fine-grained SM scheduling:** Layouts that enable fast access/dequant, plus SM-level load-balancing between W4A4/W4A8 tiles remove imbalance stalls which leads to end-to-end throughput gains vs. TRT-LLM on A100 with practical W4A4KV4 serving.