MixLLM: LLM Quantization with Global Mixed-precision between Output-features and Highly-efficient System Design

Zhen Zheng, Xiaonan Song, Chuanjie Liu @ Microsoft

Presenter: ***Taosong Fang***

The Memory Wall of LLM Inference

- **Large Memory Requirement**
  - ~100GB GPU memory v.s. ~1T parameters

How to hold it?

- **Large Memory Footprint**
  - Load ~1T parameters per decoding/single-token

How to run it efficiently?

Image from scmp.com/yp/discover/your-voice/opinion/article/3072565/elephant-room

Quantization

- **Representing data with smaller bit-width**
  - Affine w/ round and clamp: $X_q = clamp(round\left(\frac{X}{scale}\right) + zero\_point)$

- **Bit-width means precision**
  - Data round to $2^{bit-width}$ chunks
  - Larger bit-width → #chunks↑ → round-error↓ → precision↑

What to Quantize?

- **Weight only ✗**
  - ✓ Good for small-batch decoding
  - ✗ Insufficient for large-batch decoding , e.g., offline inference
  - ✗ Poor for prefill

- **Both weight and activation ✓**
  - ✓ Small-batch decoding
  - ✓ Large-batch decoding
  - ✓ Prefill-only
  - …

Let's quantize both weight and activation

What Bit-width?

- **8-bit**
  - Activation: good to preserve accuracy ✓
  - Weight: insufficient compression ✗
- **4-bit**
  - Activation: poor accuracy ✗
  - Weight: accuracy not good enough ✗

Let's use 8-bit for activation.
But how about weight?

Insight: Mixed Precision

- **Mostly 4-bit, with a small fraction of 8-bit**
  - E.g., 90% 4-bit + 10% 8-bit = 4.4-bit
- **The dimension to mix**
  - Layer-wise ✗
  - Channel-wise, a fixed per-layer fraction ✗
  - Channel-wise, global fraction rather than local fraction ✓

Let's use channel-wise mix, with global fraction

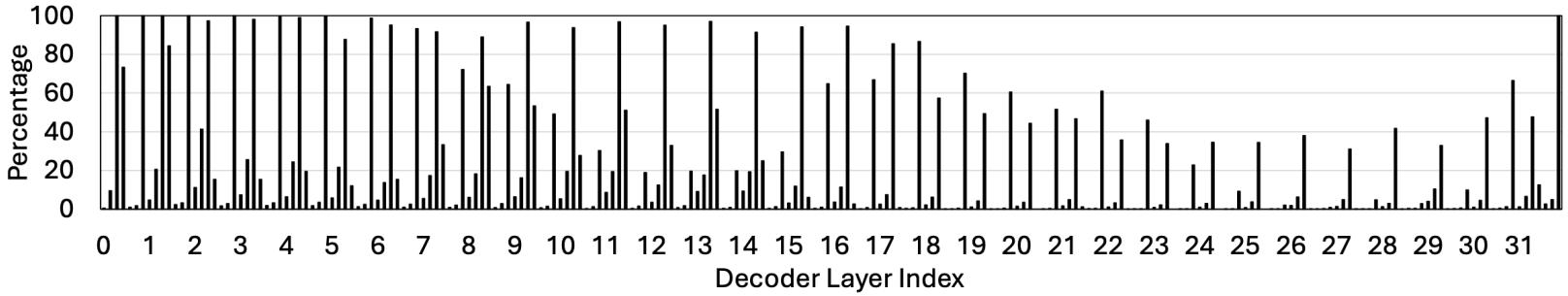
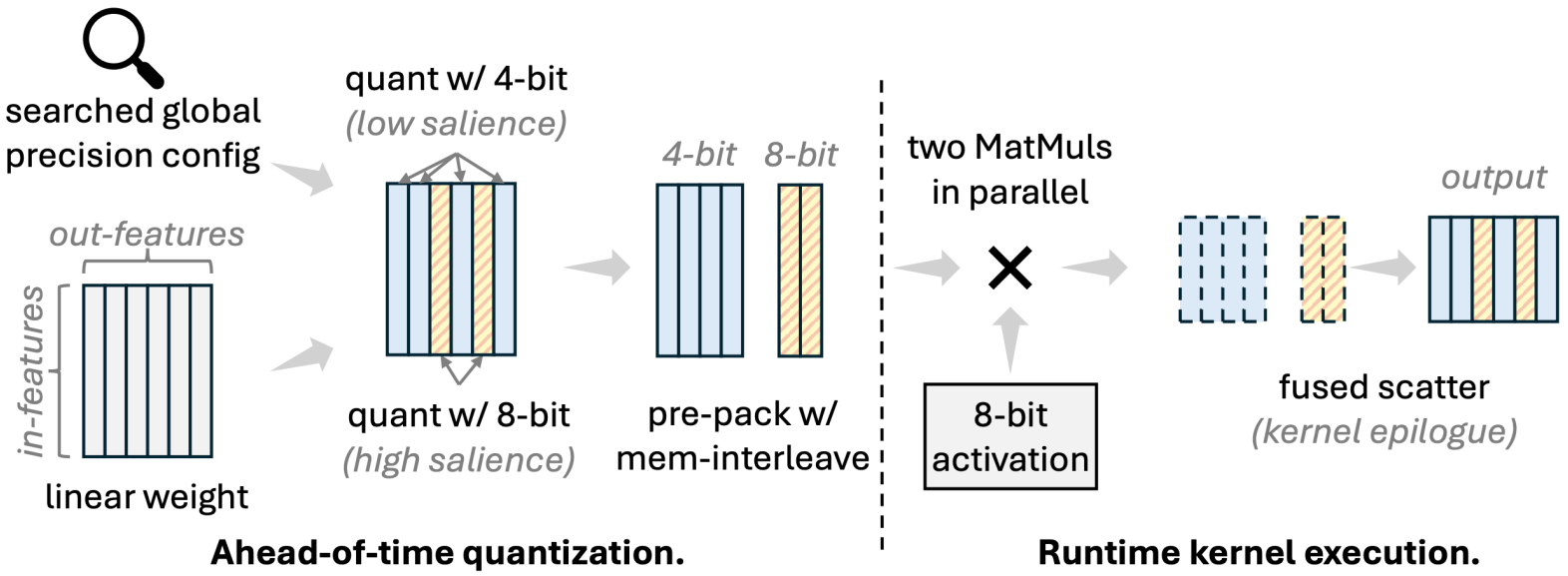


Figure 2: The percentage of high-salient out features within each linear layer of Llama 3.1 8B model according to each feature's contribution to the final loss after quantizing to 4-bit, with 10% high-salient features globally. Each decoder layer contains `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj` in order.

Insight: Mixed Precision

- **Output channel-wise: simpler for system dev**



How to Determine the Mix Config

- **Insight: the channel leads to larger end-to-end loss after quantization should have larger bit-width**
  - Turns the problem to how to determine the loss contribution of a channel's quantization

Metric: the diff of single channel's loss ➔ $S_c = \left| l(c_q) - l(c_0) \right|$

Taylor expansion approximation ➔ $l(c) \approx l(c_0) + g^T(c - c_0) + \frac{1}{2}(c - c_0)^T H(c - c_0)$

Fisher Information Matrix approximation ➔ $H \approx F = \frac{1}{|D|} \sum_{d \in D} g_d g_d^T$

The final equation ➔ $S_c = \frac{1}{|D|} \sum_{d \in D} \left| g_d^T (c_q - c_0) + \frac{1}{2} \left(g_d^T (c_q - c_0) \right)^2 \right|$

More Decisions and the Challenge

- **Challenge of group-wise w/ zero-point quant**
  - $\left[\left(W_q - z\right) * S_w\right] * \left[A_q * S_a\right]$ ✗
  - $\left(W_q - z\right) * S_w$ becomes fp16/bf16, and cannot leverage int8 Tensor Core

- **Solution: two-step dequantization**
  - Per-group $\left[\left(\boldsymbol{W_q} - \boldsymbol{z}\right) * \boldsymbol{A_q}\right] * \left[S_w * S_a\right]$
  - **Tensor Core** -> dtype cast -> multiplication
  - $\left(W_q - z\right)$ are uint4 sub, and result is within the range of int8, safe

Int to Float Can Be Slow

- **Can represent the int-to-float with a single float sub!**
  - Please refer to our paper for the details

The Software Pipeline

- **Multiple-level overlapping**
  - Global memory
  - Shared memory
  - Register
  - Tensor Core
  - SIMT Core

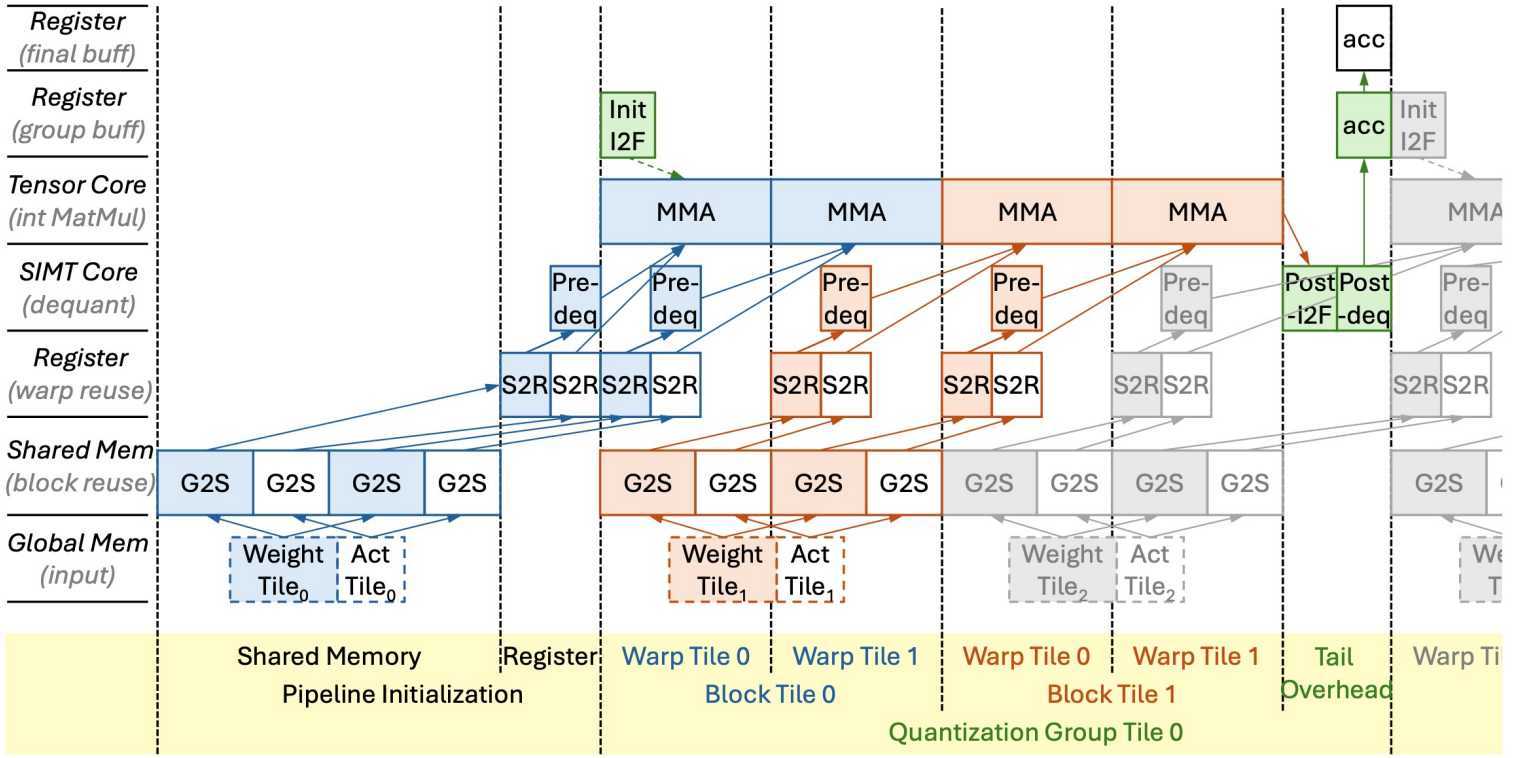


Figure 3: The GPU kernel software pipeline of group-wise W4A8/W8A8 quantized MatMul. It assumes perfect overlapping. G2S: load global to shared memory; S2R: load shared memory to register; MMA: matrix multiply-accumulation; I2F: integer to float conversion; deq: dequantize; acc: accumulate. While this pipeline is modeled on the NVIDIA A100 architecture, its fundamental principles remain applicable to subsequent generations, such as Hopper and Blackwell, subject to minor architectural refinements. For instance, newer architectures can utilize the Tensor Memory Accelerator (TMA) to load activation tensors directly, bypassing registers before they reach the Tensor Cores. Furthermore, these architectures support warp specialization for memory loading as an alternative to a uniform execution scheme.

Good Accuracy

- **Nearly-lossless with W4.4A8**

Table 2: Downstream tasks evaluation (↑) on Llama-3.1-8B/Qwen2.5-7B/Mistral-7B-v0.3. The above is the average of the three models. BBH is 3 shot, MMLU pro is 5 shot, and others are zero shot.

	BBH	GPQA	MMLU-Pro	MuSR	ARCc	HellaSwag
float16	48.62 46.52/54.09/45.25	30.86 31.08/33.11/28.39	35.52 32.91/43.86/29.80	41.07 37.99/44.51/40.72	52.24 53.41/51.02/52.30	79.43 78.92/78.94/80.43
SmoothQuant W8A8	47.82 46.37/52.57/44.52	30.90 31.40/33.94/27.36	35.04 32.61/42.98/29.52	42.06 39.05/46.39/40.73	51.74 53.33/50.00/51.88	79.20 78.88/78.48/80.24
QuaRot W4A4	41.10 36.96/45.42/40.92	27.53 25.41/28.94/28.23	27.60 22.99/34.40/25.42	39.46 37.92/40.68/39.77	45.99 43.00/46.33/48.63	74.85 72.87/73.54/78.14
QuaRot W4A8	46.95 44.95/52.98/42.92	30.28 30.96/30.71/29.18	33.60 29.95/42.45/28.41	41.65 39.05/45.58/40.32	51.39 50.00/52.30/51.88	78.55 77.83/77.84/79.98
QServe W4A8	45.78 40.98/51.23/45.14	30.02 28.99/32.50/28.56	32.84 28.16/41.72/28.63	39.92 37.60/41.59/40.57	50.54 51.28/49.15/51.19	78.10 76.90/77.52/79.89
MixLLM W4A8	46.92 43.44/44.75/52.59	29.90 29.58/28.26/31.87	33.75 30.18/29.59/41.49	41.70 38.81/43.11/43.19	51.82 51.71/51.88/51.88	78.61 77.94/79.71/78.17
MixLLM W4.4A8	48.17 46.27/52.58/45.66	30.09 29.17/31.75/29.36	34.53 31.08/43.26/29.26	41.74 39.32/44.79/41.11	52.70 53.67/51.96/52.47	79.00 78.20/78.58/80.21
MixLLM W8A8	48.84 46.84/54.35/45.34	30.93 30.51/33.21/29.07	35.54 33.00/43.80/29.83	40.94 37.32/44.91/40.59	52.10 53.24/50.94/52.13	79.42 78.98/78.88/80.40

Good Efficiency

- **SOTA kernel efficiency**

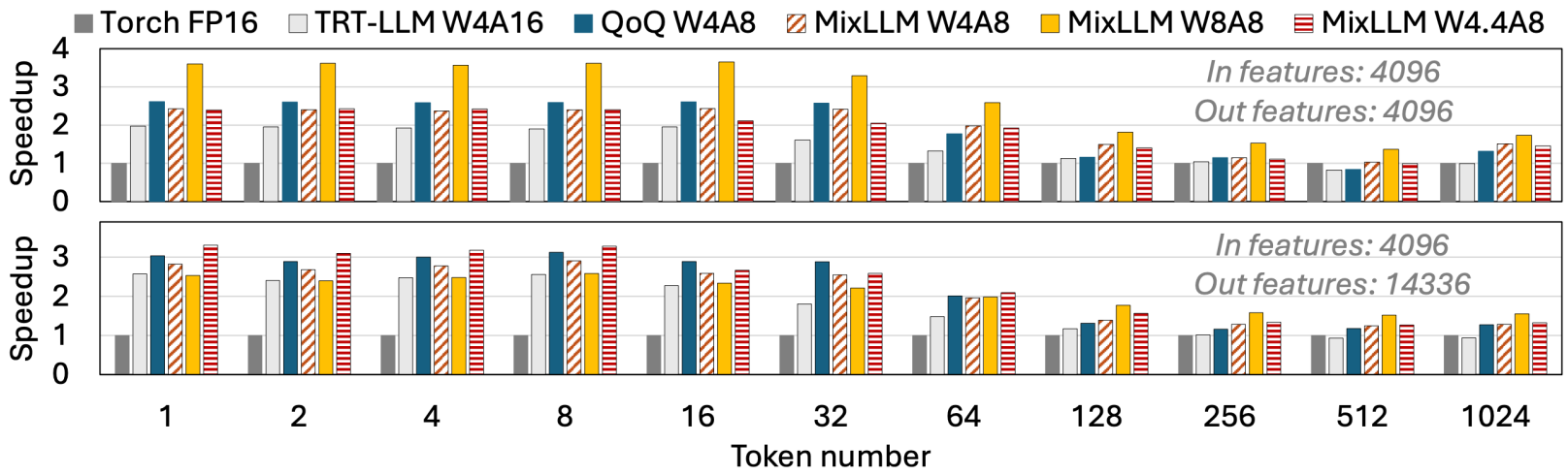


Figure 5: The speedup of two types of single linear layers over torch FP16 baseline on A100 GPU.

THANKS

https://github.com/microsoft/MixLLM