AWQ: Activation-Aware Weight Quantization for on-device LLM Compression and Acceleration

MLSys 2024

Presenter: Raunak Shah
CS598 Fall 2024

BACKGROUND

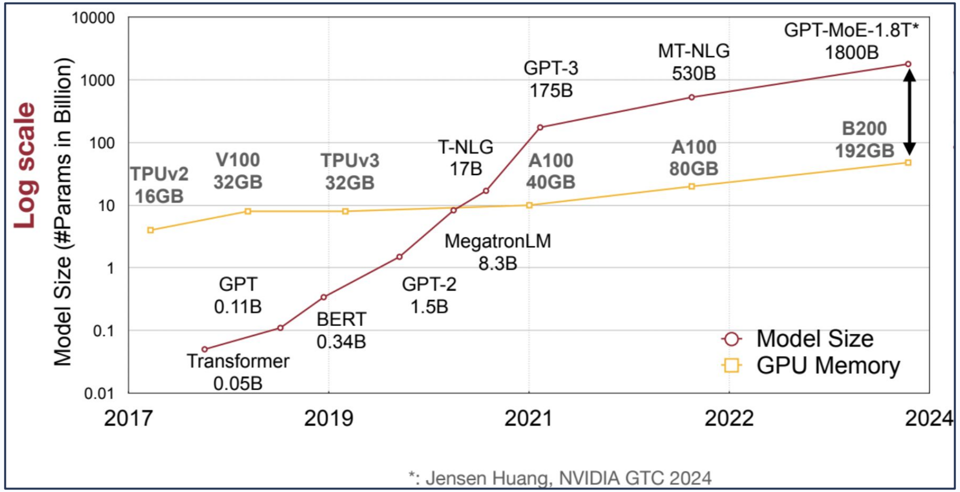
Deploying LLMs directly on edge devices is useful

- No costs from centralized cloud infrastructure
- No latency of sending data to cloud server
- Improved data security and privacy
- Constraints
  - Less resources
  - Low power devices
  - May not always have access to the internet

BACKGROUND

Deploying LLMs directly on edge devices is useful

- No costs from centralized cloud infrastructure
- No latency of sending data to cloud server
- Improved data security and privacy



But it is a hard problem to solve

- Large model sizes
- High LLM serving costs

RELATED WORK

General Quantization Methods

- Quantization-aware Training (QAT)
  - Uses backprop to update quantized weights, does not scale well for LLMs
- **Post-Training Quantization (PTQ)**
  - Training free

Quantizing LLMs

- W8A8 - quantize both activations and weights to INT8
  - SmoothQuant
- **W4A16 - only weights are quantized to 4-bits**
  - RTN - round to nearest
  - GPTQ

RELATED WORK

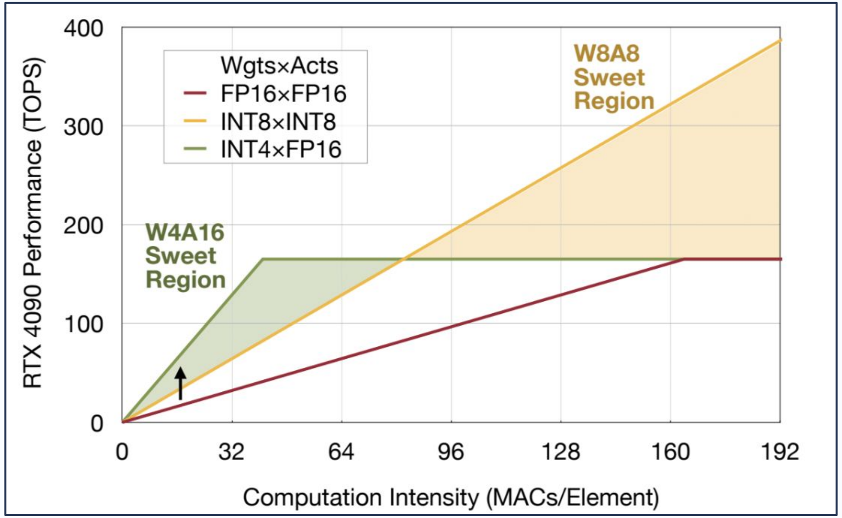
- **RTN**
  - Vanilla baseline
  - Directly round each weight to the nearest value in the 4-bit scale
- **GPTQ**
  - Greedy, layer-by-layer quantization approach
  - For each layer, it minimizes the error in the output of that layer
  - Iteratively quantizes weights while trying to preserve the layer's original behavior
  - Uses Hessian-based information to determine quantization order

RELATED WORK

- **RTN**
  - Significant quantization error
- **GPTQ**
  - Quantizes weights in a specific order (greedy approach)
  - For some models the standard order does not work well (reordering required)
  - Uses a calibration dataset to optimize its quantization
  - When minimizing error on this dataset, it may overfit to those specific examples

QUANTIZATION

- Low-bit weight quantization can reduce LLM inference costs
- Map a higher-precision floating point number into a lower-precision float or int

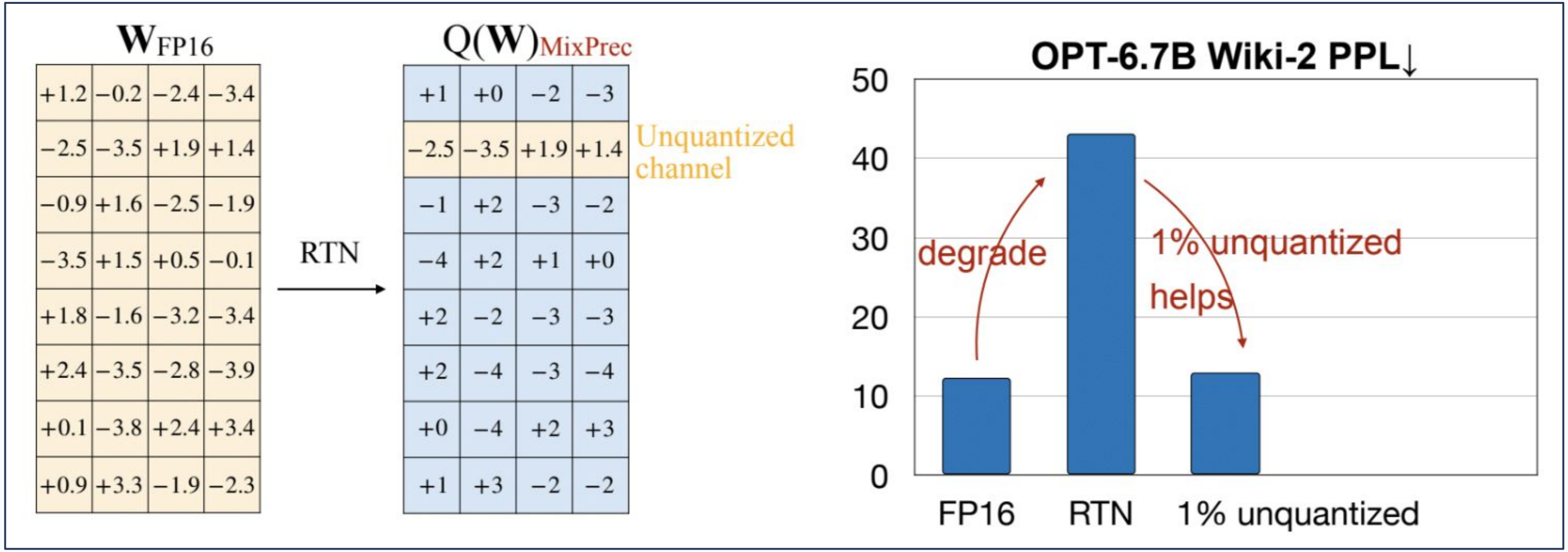


LLMs on the edge are memory-bound: W8A8 quantization is not enough

KEY OBSERVATIONS

Context

- The weights of LLMs are not equally important
- Some **salient** weights contribute more to performance
  - If we intentionally do not quantize them, we can improve performance



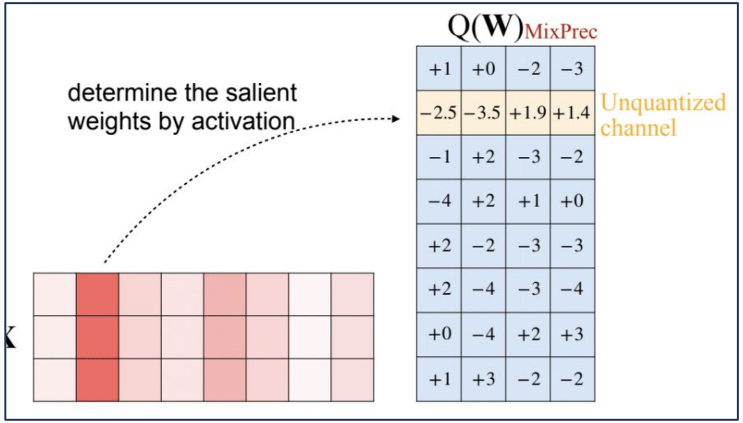
KEY OBSERVATIONS

Context

- The weights of LLMs are not equally important
- Some **salient** weights contribute more to performance
    - If we intentionally do not quantize them, we can improve performance

But how to know which weights are important?

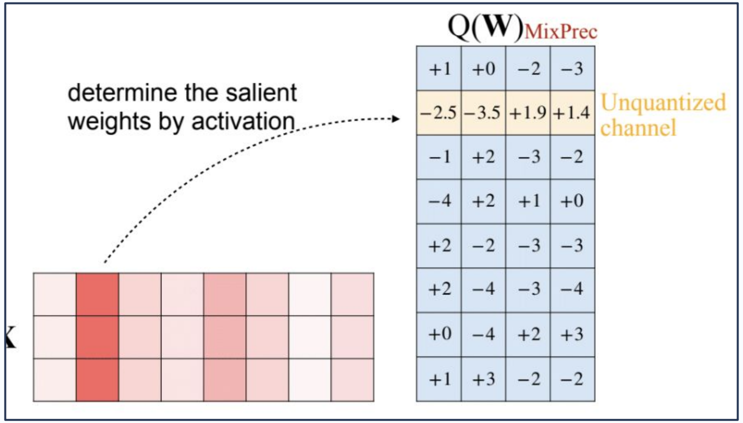
- Common approach - look at L2 norm
- Selecting weights based on magnitude of activations



KEY OBSERVATIONS

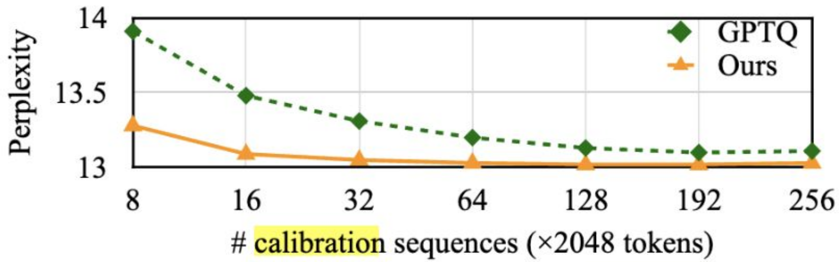
New problem

- Using a mixed-precision weight datatype makes a system implementation on the edge difficult



AWQ: CALIBRATION SET

- GPTQ optimizes over a calibration set to change the weights slightly for optimal performance
    - It is easy for models to overfit to this calibration set
- AWQ uses the calibration set only to identify which weights are important for activations
    - Doesn't directly optimize weights to match outputs on this data



(a) Our method needs a smaller calibration set

Calib \ Eval	GPTQ PubMed	GPTQ Enron	Ours PubMed	Ours Enron
PubMed	32.48	50.41 +4.89	32.56	45.07 +0.50
Enron	+2.33 34.81	45.52	+0.60 33.16	44.57

(b) Our method is more robust to calibration set distribution

AWQ: HOW TO AVOID MIXED PRECISION?

- Define the quantization function as:

$$Q(\mathbf{w}) = \Delta \cdot \text{round}(\frac{\mathbf{w}}{\Delta}), \Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}}.$$

- N = quantization bits, w = weight, Δ = quantization scalar

AWQ: PER-CHANNEL SCALING

- Define the quantization function as:

$$Q(\mathbf{w}) = \Delta \cdot \text{round}(\frac{\mathbf{w}}{\Delta}), \Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}}.$$

- N = quantization bits, w = weight, Δ = quantization scalar
- Try scaling the weight by a factor of s. To keep the result the same, we need to scale down the activations by the same factor. Then **wx** will look like:

fuse to previous op

$$\mathbf{WX} \xrightarrow{\text{Quantize}} Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X})$$

- Or equivalently:

$$Q(w \cdot s) \cdot \frac{x}{s} = \Delta' \cdot \text{Round}(\frac{ws}{\Delta'}) \cdot x \cdot \frac{1}{s},$$

AWQ: PER-CHANNEL SCALING

$$Q(\mathbf{w}) = \Delta \cdot \text{round}(\frac{\mathbf{w}}{\Delta}), \Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}}.$$

Quantization scaler, Absmax within the group

$$Q(w) \cdot x = \Delta \cdot \text{Round}(\frac{w}{\Delta}) \cdot x \quad \rightarrow \quad \text{Err}(Q(w) \cdot x) = \Delta \cdot \text{Err}(\text{Round}(\frac{w}{\Delta})) \cdot x$$

$$\approx$$

$$Q(w \cdot s) \cdot \frac{x}{s} = \Delta' \cdot \text{Round}(\frac{w \cdot s}{\Delta'}) \cdot x \cdot \frac{1}{s} \qquad \text{Err}(Q(w \cdot s) \cdot \frac{x}{s}) = \Delta' \cdot \text{Err}(\text{Round}(\frac{w \cdot s}{\Delta'})) \cdot x \cdot \frac{1}{s}$$

A scalar with an expectation of 0.25

- Now round() has a fixed expected value (0.25) as it is bounded between 0 and 0.5
- The Δ values do not change, since **usually** scaling a single channel does not change the global absolute maximum weight
- ⇒ As a result, effective quantization error is **reduced by a factor of s**

AWQ: PER-CHANNEL SCALING

$$\Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}}$$

OPT-6.7B	$s=1$	$s=1.25$	$s=1.5$	$s=2$	$s=4$
proportion of $\Delta' \neq \Delta$	0%	2.8%	4.4%	8.2%	21.2%
average Δ'/Δ	1	1.005	1.013	1.038	1.213
average $\frac{\Delta'}{\Delta} \cdot \frac{1}{s}$	1	0.804	0.676	0.519	**0.303**
Wiki-2 PPL	23.54	12.87	12.48	**11.92**	12.36

- Reduction in quantization error relies on the assumption that Δ will stay the same
- When does this not happen?
  - Higher scale values change Δ more, because there is a higher chance that the absolute global maximum weight will change
  - **Changing the value of Δ too much increases the error for other (non-salient) channels**
  - ⇒ So we need to choose the scale carefully to achieve optimal perplexity

AWQ: PER-CHANNEL SCALING

- Which scale values to use? Optimize the quantization error as a function of s:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} \mathcal{L}(\mathbf{s})$$
$$\mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \text{diag}(\mathbf{s}))(\text{diag}(\mathbf{s})^{-1} \cdot \mathbf{X}) - \mathbf{W}\mathbf{X}\|$$

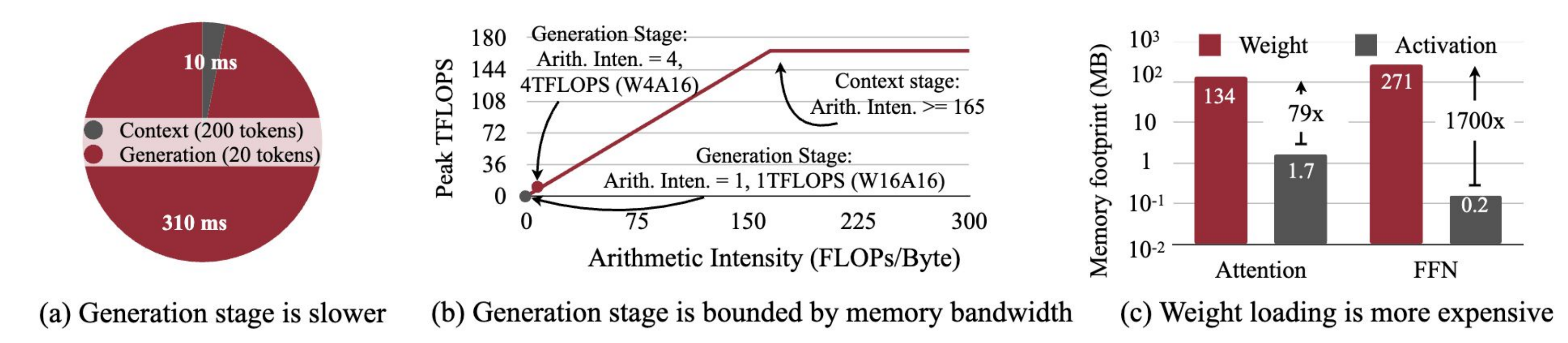
- Here **s** represents a list of scaling factors for each channel
- Define a search space to make solving this easier:

$$\mathbf{s} = \mathbf{s_X}^{\alpha}, \quad \alpha^* = \arg\min_{\alpha} \mathcal{L}(\mathbf{s_X}^{\alpha})$$

$\mathbf{s_X}$ = avg magnitude of activation (per channel), α = hyperparameter between [0, 1]

AWQ: PRACTICAL IMPLICATIONS

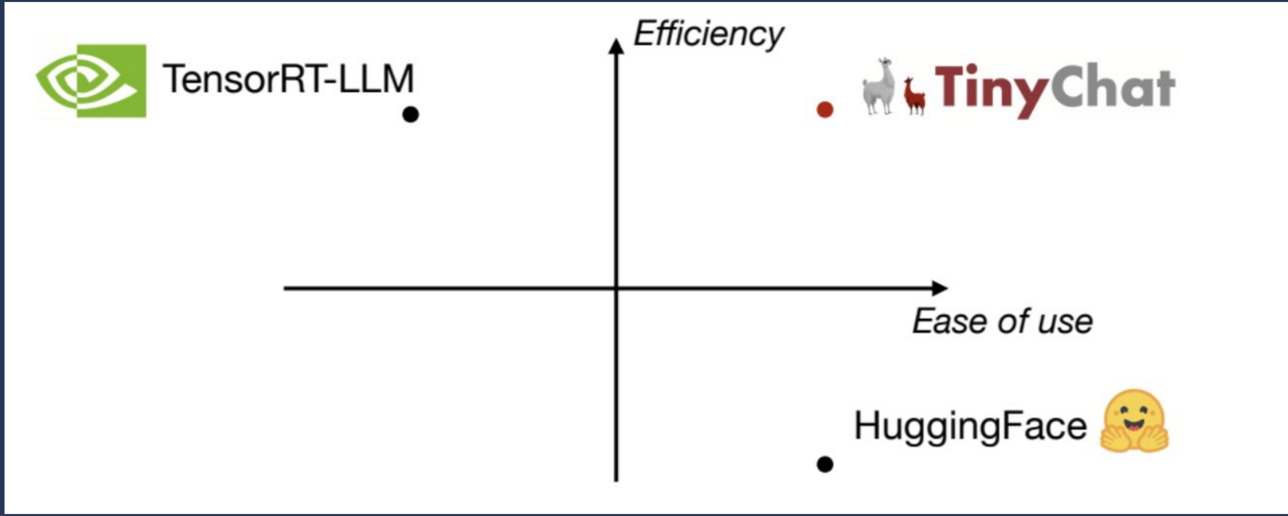
- (a) The **generation stage** is much slower than the context stage (e.g. generating tokens vs summarization)
- (b) This generation stage for standard FP16 LLMs is **memory-bound**
  - Why? Empirically, ratio of FLOPs : memory access is ~ 1 whereas peak GPU throughput is 165 TFLOPs
- (c) For memory accesses, **weight memory >>> activation memory**
- **AWQ reduces the weight memory by 4x** (16 bit -> 4 bit)



Bottleneck analysis for Llama-2-7B on NVIDIA RTX 4090

TINYCHAT FOR ON-DEVICE LLMS

- 4-bit weight quantization can lead to 4x speedup in performance (theoretically)
- But there are practical considerations involved as well
- TinyChat - a system for LLM Inference using AWQ
  - Over 3x speedup compared to standard HuggingFace FP16 implementation



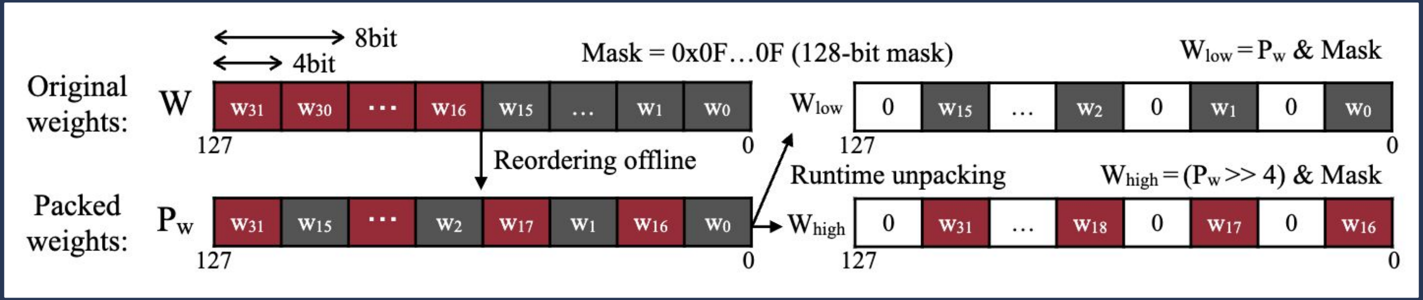
TINYCHAT FOR ON-DEVICE LLMS

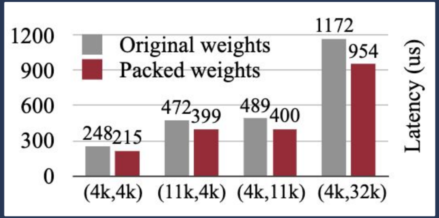
- Methods like SmoothQuant (W8A8) have the same data precision for storage and computation, which lets kernels be simple
- For methods like AWQ (W4A16), dequantization must be added to the GPU kernels for optimal performance, which poses implementation challenges

- **On-the-fly weight dequantization:**
  - Need to dequantize INT4 to FP16 before performing matrix multiplication.
  - Fuse kernels for dequantization and matrix multiplication together.

TINYCHAT FOR ON-DEVICE LLMS

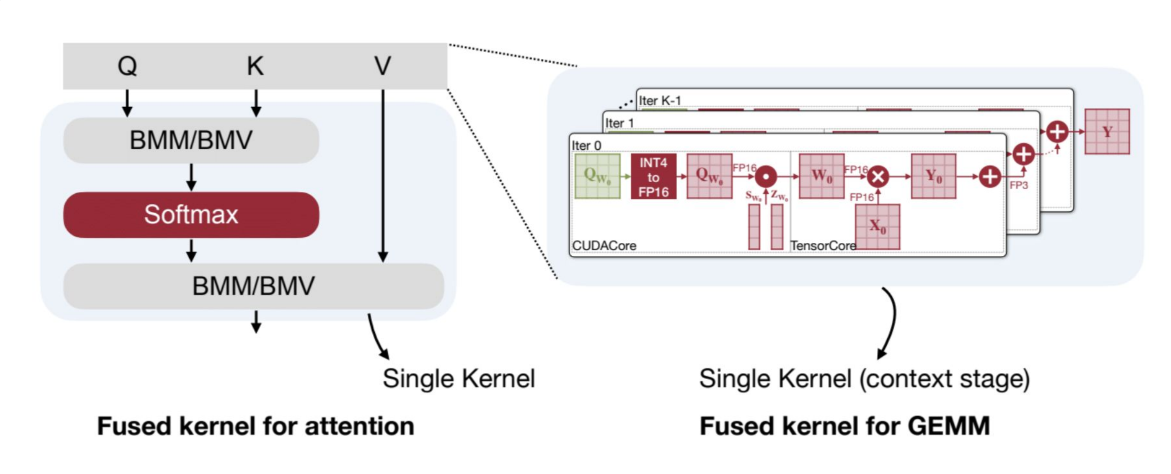
- **SIMD-Aware Weight packing:**
  - Dequantizing a single 4-bit weight still involves 1 shift, 1 bitwise AND, and 1 FMA scaling operation
  - Change the bit-packing so that **all** the weights can be dequantized using only 3 SIMD instructions





TINYCHAT FOR ON-DEVICE LLMS

- **Kernel fusion:**
  - Reducing the number of kernel calls reduces DRAM accesses, speeding up compute
  - Fuse all operators (e.g. multiplication, division, square root) into a single kernel.
  - Attention layers:
    - Fuse QKV projections into a single kernel
    - Perform on-the-fly positional embedding calculation.
    - Preallocate KV caches and perform cache updates within the attention kernel.



EVALUATION

- **Benchmarking done on:**
  - LLaMa and OPT family of LLM models
  - Vicuna (instruction-tuned model)
  - OpenFlamingo-9B, LLaVA-13B (visual models)
- **Dataset/Evaluation**
  - LM tasks from WikiText2, metric - perplexity
- **Baselines**
  - Round-to-Nearest (RTN)
  - GPTQ
  - Disregard baselines that use backprop/regression to make quantized weights more accurate

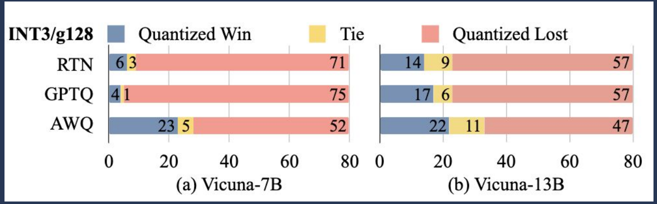
EVALUATION

PPL↓		Llama-2			LLaMA			
		7B	13B	70B	7B	13B	30B	65B
FP16	-	5.47	4.88	3.32	5.68	5.09	4.10	3.53
INT3 g128	RTN	6.66	5.52	3.98	7.01	5.88	4.88	4.24
	GPTQ	6.43	5.48	3.88	8.81	5.66	4.88	4.17
	GPTQ-R	6.42	5.41	3.86	6.53	5.64	4.74	4.21
	AWQ	**6.24**	**5.32**	**3.74**	**6.35**	**5.52**	**4.61**	**3.95**
INT4 g128	RTN	5.73	4.98	3.46	5.96	5.25	4.23	3.67
	GPTQ	5.69	4.98	3.42	6.22	5.23	4.24	3.66
	GPTQ-R	5.63	4.99	3.43	5.83	5.20	4.22	3.66
	AWQ	**5.60**	**4.97**	**3.41**	**5.78**	**5.19**	**4.21**	**3.62**

Comparison with other models on LLaMa

Wikitext2 PPL↓	Mixtral-8x7B	Mistral-7B
FP16	5.94	4.14
INT4-g128	6.05	4.30
INT3-g128	6.52	4.83

Mistral perplexity scores
No comparison provided



Instruction-tuned models

MBPP (7B)	pass@1	pass@10
FP16	38.53	49.77
RTN	37.51	48.49
GPTQ	31.97	44.75
AWQ	**40.64**	**49.25**

GSM8K	7B	13B	70B
FP16	13.87	26.16	56.41
RTN	11.07	21.23	53.98
GPTQ	12.13	24.26	56.03
AWQ	**13.57**	**25.25**	**56.40**

Programming and math tasks

EVALUATION

COCO (CIDEr ↑)		0-shot	4-shot	8-shot	16-shot	32-shot	*Δ(32-shot)*
FP16	-	63.73	72.18	76.95	79.74	81.70	-
INT4 g128	RTN	60.24	68.07	72.46	74.09	77.13	-4.57
	GPTQ	59.72	67.68	72.53	74.98	74.98	-6.72
	AWQ	**62.57**	**71.02**	**74.75**	**78.23**	**80.53**	**-1.17**
INT3 g128	RTN	46.07	55.13	60.46	63.21	64.79	-16.91
	GPTQ	29.84	50.77	56.55	60.54	64.77	-16.93
	AWQ	**56.33**	**64.73**	**68.79**	**72.86**	**74.47**	**-7.23**

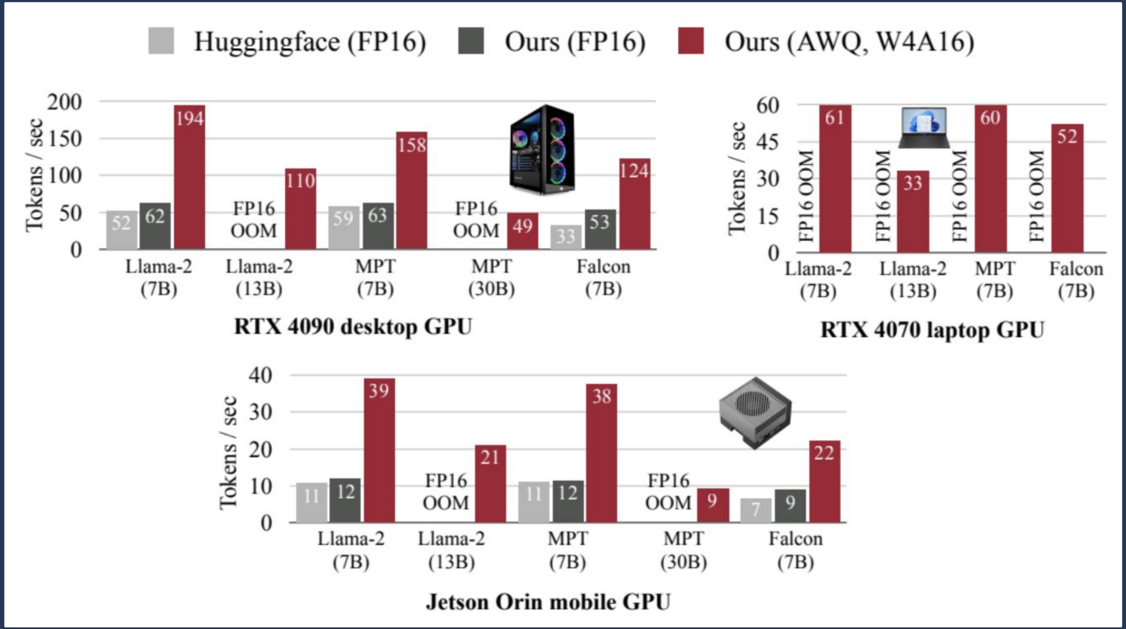
Perplexity results on visual language models

OPT (Wiki PPL↓)	1.3B	2.7B	6.7B	13B	30B
FP16	14.62	12.47	10.86	10.13	9.56
RTN	10476	193210	7622	17564	8170
GPTQ	46.67	28.15	16.65	16.74	11.75
AWQ +GPTQ	**35.71**	**25.70**	**15.71**	**13.25**	**11.38**

Extreme low-bit (INT2) quantization

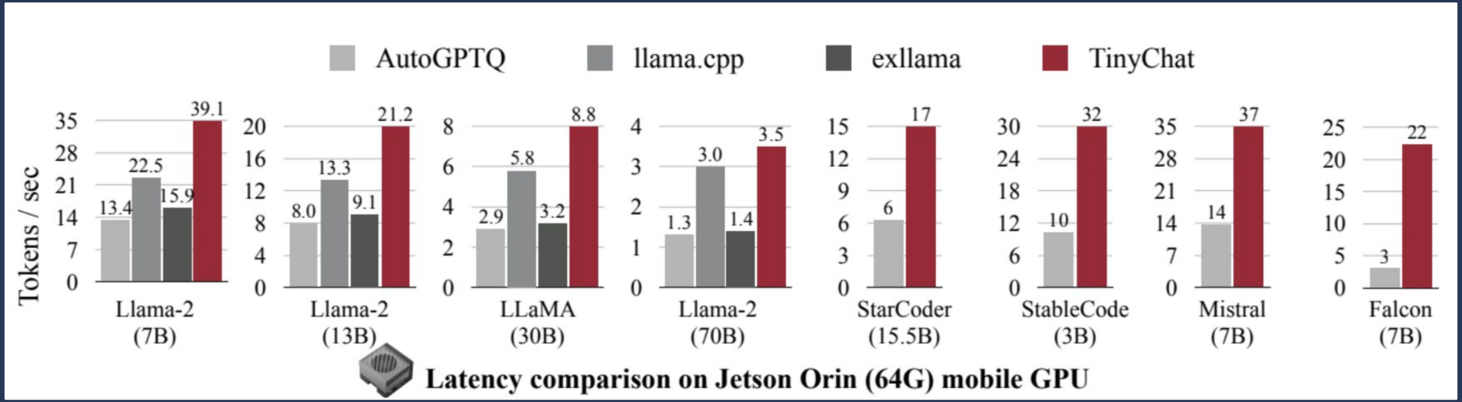
EVALUATION

- Over 3x faster than HuggingFace FP16 implementation



EVALUATION

- Faster than other LLM inference systems



EVALUATION

<u>Question</u>: Can you explain this meme in detail?
<u>INT4-g128 RTN</u>: The meme in the image features a plate of food, likely fried chicken, with a unique twist. There are small pictures of the Earth and other planets placed on top of the food, creating an interesting and amusing visual. Alongside the plate, a caption reads, "Sometimes I look at pictures of the Earth from space and I marvel at how beautiful it all is." This combination of food and planet images adds a touch of humor and curiosity, making the viewer wonder about the context and intent behind the meme.
<u>INT4-g128 AWQ</u>: The meme in the image is a lighthearted and humorous take on the concept of looking at pictures of the Earth from space. A plate of fried food, specifically chicken nuggets, is presented with the caption, "Sometimes I look at pictures of the Earth from space and I marvel at how beautiful it all is." The playful combination of a food item and an Earth-related context creates a fun and entertaining visual.

W4-RTN: A model airplane flying in the sky.
W4-AWQ: Two toy airplanes sit on a grass field.

W4-RTN: A man is holding a baby elephant in his arms.
W4-AWQ: A man and his daughter pose with an elephant.

W4-RTN: A man and a dog walking past some bushes.
W4-AWQ: Two dogs are walking on the street.

IMPACT

- AWQ models downloaded over 1 million times on HuggingFace

NVIDIA	TensorRT-LLM	https://github.com/NVIDIA/TensorRT-LLM#key-features		Transformer Quantization API	https://huggingface.co/docs/transformers/main_classes/quantization
IBM	Granite	IBM's internal code model, Granite, utilizes AWQ for quantization.		lmdeploy	https://github.com/InternLM/lmdeploy/blob/main/lmdeploy/lite/quantization/awq.py
Berkeley UNIVERSITY OF CALIFORNIA	vLLM	https://github.com/vllm-project/vllm/blob/main/vllm/model_executor/layers/quantization/awq.py	Google Cloud	Vertex AI	https://console.cloud.google.com/vertex-ai/publishers/google/model-garden
	lm-sys/FastChat	https://github.com/lm-sys/FastChat/blob/main/docs/awq.md	intel.		https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md

IMPACT

- Can be used for LLM deployment on small edge devices (NVIDIA Jetson Orin Nano)
- 7 GB memory
- 7B parameters

STRENGTHS, WEAKNESSES, THOUGHTS

- MLSys 2024 Best Paper Award
- Idea is simple and practical, used extensively by real users
- Comprehensive evaluation

STRENGTHS, WEAKNESSES, THOUGHTS

- MLSys 2024 Best Paper Award
- Idea is very simple and practical, used extensively by real users
- Comprehensive evaluation

- Only 2 main baselines (one is vanilla rounding), but the field is newer so it makes sense
- The need to compare across different GPU sizes is unclear since the framework should still work the same either way

STRENGTHS, WEAKNESSES, THOUGHTS

- Positioning the paper as ‘Quantization for Edge Devices’ may be a stretch
  - Original version on arxiv only had the quantization framework
  - Other low-bit quantized models already existed, which could technically also be deployed on edge devices (just with lower performance)
  - Kernel-level optimizations are not specific to edge devices, required for practical use

- Future ideas:
  - Support both activation and weight quantization in a low-bit setting
  - Further reduce dependency on calibration set
  - Adapt quantization methods depending on the model architecture (transformer, CNN, etc)

Questions?

REFERENCES

- Github: https://github.com/mit-han-lab/llm-awq
- Official Slides: https://www.dropbox.com/scl/fi/dtnp6h6y1mnp7g036axu6/AWQ-slide.pdf?rlkey=ffgh50hxhx8dmsnjiu8kef0ou&dl=0
- Paper: https://arxiv.org/pdf/2306.00978
- Video: https://www.youtube.com/watch?v=3dYLj9vjfA0