AWQ: Activation-Aware Weight Quantization for On-Device LLM Compression and Acceleration

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen,
Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, Song Han

Presented By:

Paul-Andrei Aldea, Michael Hwang, John Kim, Jason Liang

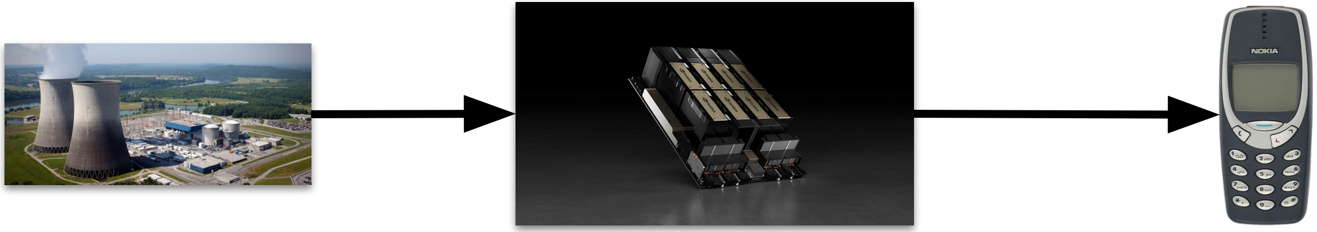
Problem & Motivation

Desire to run large models on-device (rather than communicate remotely).

- **Benefits:** reduced latency, offline availability, privacy improvement, etc.

Models are too memory-intensive to fit on edge-devices (e.g. mobile phones):

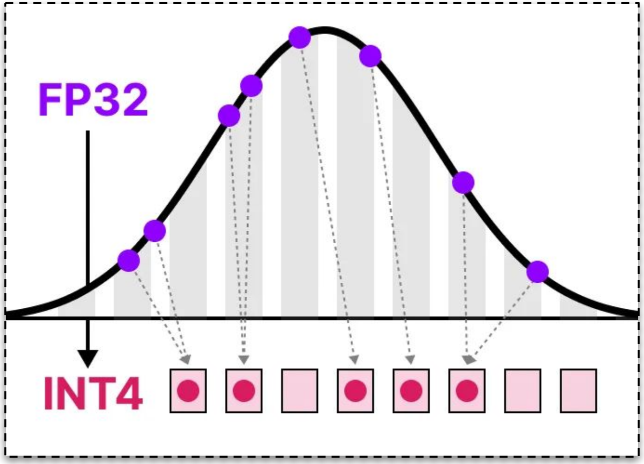
- GPT-3 @ 175B parameters → 350GB for weights in FP16 (2B per weight).
- High-end GPUs (H100s - now B200s) do not exceed ~96GB of memory.
- Low-end GPUs (RTX 4070s) have ~12GB of memory.



Definitions and Intuition

Quantization: Reduces the precision of numerical values in a model. Typically used to reduce model size (e.g. [FP16] weights → [INT4] weights).

- Trade model accuracy for decreased memory usage and computation time.



- GPT-3 @ 175B parameters:
  - ~ 87.5 GB in [INT4].
  - ~ 65.6 GB in [INT3].

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Quantization

$$Q(\mathbf{w}) = \boxed{\Delta} \cdot \boxed{\text{Round}\left(\frac{\mathbf{w}}{\Delta}\right)}, \quad \Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}}$$

Recover approximate weights with delta factor.

Store parameter values as this (e.g, INT4).

https://www.dropbox.com/scl/fi/dtnp6h6y1mnp7g036axu6/AWQ-slide.pdf?rlkey=ffgh50hxhx8dmsnjiu8kef0ou&e=1&dl=0,
https://leimao.github.io/article/Neural-Networks-Quantization/

Quantization

FP16 $$w = \begin{bmatrix} -6.1 & -5.8 & -4.1 & 2.6 & 0.6 \end{bmatrix} \quad \Delta = \frac{6.1 \cdot 2}{2^3} = 1.525$$

$$\frac{w}{\Delta} = \begin{bmatrix} -4. & -3.803 & -2.689 & 1.705 & 0.393 \end{bmatrix}$$

$\downarrow$ Rounding

INT3 $$\text{Round}(\frac{w}{\Delta}) = \begin{bmatrix} -4 & -4 & -3 & 2 & 0 \end{bmatrix}$$

$\downarrow$ Recovery

$$\Delta \cdot \text{Round}(\frac{w}{\Delta}) = \begin{bmatrix} -6.1 & -6.1 & -4.575 & 3.05 & 0. \end{bmatrix}$$ FP16

Reference: Initial test example (not shown) produced using ChatGPT; the current example (shown) was produced with numpy

Definitions and Intuition

Q: Are there weights which are more relevant for the predictions of a model?

Idea: Consider Activation Magnitude (i.e., Look for “Massive Activations”)

- How strongly a weight responds to an input might indicate importance.
- Only preserve weights in higher precision which are “strongly activated.”

Leverage a calibration dataset to determine salient model weights.

Quantization (% and type) vs Perplexity (↓) Overview

PPL ↓	FP16	RTN (w3-g128)	FP16% (based on act.)			FP16% (based on W)			FP16% (random)		
			0.1%	1%	3%	0.1%	1%	3%	0.1%	1%	3%
OPT-1.3B	14.62	119.00	25.03	16.91	16.68	108.71	98.55	98.08	119.76	109.38	61.49
OPT-6.7B	10.86	23.54	11.58	11.39	11.36	23.41	22.37	22.45	23.54	24.23	24.22
OPT-13B	10.13	46.04	10.51	10.43	10.42	46.07	48.96	54.49	44.87	42.00	39.71

Table 1. Keeping a small fraction of weights (0.1%-1%) in FP16 significantly improves the performance of the quantized models over round-to-nearest (RTN). It is only effective when we select the important weights in FP16 by looking at *activation* distribution instead of *weight* distribution. We highlight results with a decent perplexity in green. We used INT3 quantization with a group size of 128 and measured the WikiText perplexity (↓).

Perplexity achieved with no quantization.
* baseline *

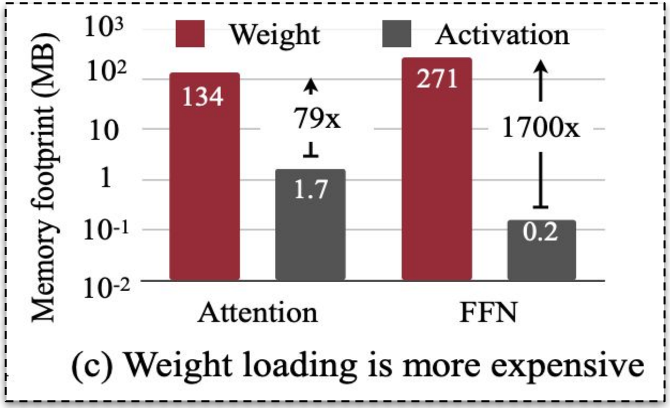
Keeping 1% of "massively activated" weights shows comparable performance.

Keeping 1% of large magnitude weights is not much better than random quantization.

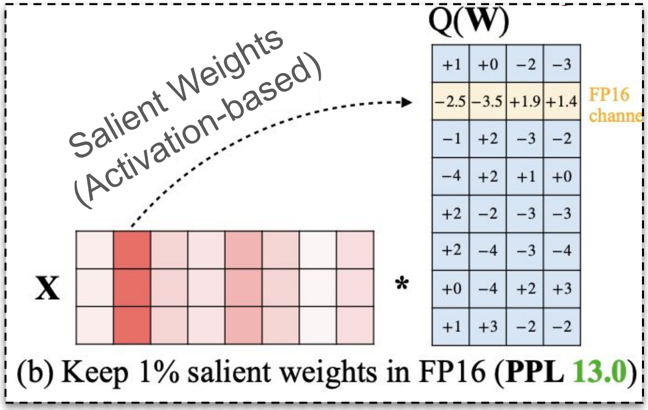
Definitions and Intuition

Reduce the precision of **non-salient** weights, and keep the precision of **salient** weights.

- Goal: Reduce model size, while maintaining comparable performance



Memory Footprint of Layers and Values



Activation Magnitude-aware Quantization.

Hardware Limitations and Workarounds

Common ISAs do not have instructions for mixed-precision arithmetic.

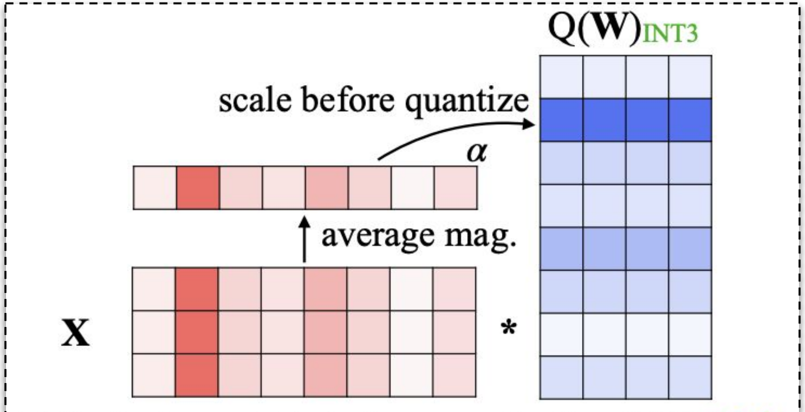
- For example, cannot compute [INT3] x [FP16].
- Storing weights in mixed precision is also hardware inefficient.

Idea: Keep all weights in low precision and improve recovery for salient weights.

AWQ Algorithm

AWQ quantizes all weights, mitigating hardware inefficiencies.

- Initially scale salient weights to improve later recovery.
- Inversely scale inputs to maintain equivalent computation.



(c) Scale the weights before quantization (**PPL** 13.0)

AWQ Algorithm

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

Given an input ***x***, the error for approximating ***w x*** by Q(***w***) ***x*** is as follows:

$$w \cdot x - \Delta \cdot \text{Round}(\frac{w}{\Delta}) \cdot x = \Delta \cdot \left(\frac{w}{\Delta} - \text{Round}(\frac{w}{\Delta})\right) \cdot x$$
$$= \Delta \cdot \text{RoundError}(\frac{w}{\Delta}) \cdot x$$

How can we do better?

https://www.dropbox.com/scl/fi/dtnp6h6y1mnp7g036axu6/AWQ-slide.pdf?rlkey=ffgh50hxhx8dmsnjiu8kef0ou&e=1&dl=0

Scaling Trick

$$\mathrm{Round}(w) \quad \text{vs} \quad \frac{1}{1.5}\mathrm{Round}(1.5w)$$

9 $\qquad$ $\boldsymbol{w} = 8.5$ $\qquad$ 8.667

Assuming that the rounding error is similar, the latter has higher output granularity and gives a better estimate of $\boldsymbol{w}$.

In fact, assuming equal rounding error, the latter is smaller by factor of 1/1.5.

AWQ Algorithm

Given a set of weights, consider scaling a selected scalar entry ***w*** by ***s*** > 1.

This gives new **Δ’** and new estimate $w \cdot x \approx \Delta' \cdot \mathrm{Round}(\frac{s \cdot w}{\Delta'}) \cdot \frac{x}{s}$

Error is $\Delta' \cdot \mathrm{RoundError}(\frac{s \cdot w}{\Delta'}) \cdot \frac{x}{s}$ instead of $\Delta \cdot \mathrm{RoundError}(\frac{w}{\Delta}) \cdot x$

Assuming that **Δ’** is roughly equal to **Δ** and that RoundErrors are roughly equal (empirically true): new error is is roughly 1/***s*** of the old one!

https://www.dropbox.com/scl/fi/dtnp6h6y1mnp7g036axu6/AWQ-slide.pdf?rlkey=ffgh50hxhx8dmsnjiu8kef0ou&e=1&dl=0

AWQ Algorithm

Compute saliency of weights (i.e., “average magnitude of activation”) using a calibration dataset (e.g., “The Pile” dataset)

Loss function: $\mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathrm{diag}(\mathbf{s}))(\mathrm{diag}(\mathbf{s})^{-1} \cdot \mathbf{X}) - \mathbf{W}\mathbf{X}\|$

Here channel **i** is scaled by some scalar **s(i)**

- Use larger **s(i)** for more salient weights

Perform a block search between no scaling vs. aggressive scaling.

https://www.dropbox.com/scl/fi/dtnp6h6y1mnp7g036axu6/AWQ-slide.pdf?rlkey=ffgh50hxhx8dmsnjiu8kef0ou&e=1&dl=0

Key Baseline for Evaluation: GPTQ

GPTQ extends another previous work, OBQ

Informal Idea: Greedily quantize weights, opting for ones with lower weighted quantization errors

GPTQ makes clever algorithmic insights and edits to OBQ

- e.g., “Lazy Batch-Updates”, “Cholesky Reformulation”
- Newer approach uses a “reordering trick”: GPTQ-Reorder

Potential problem: overfitting during calibration

https://arxiv.org/pdf/2210.17323

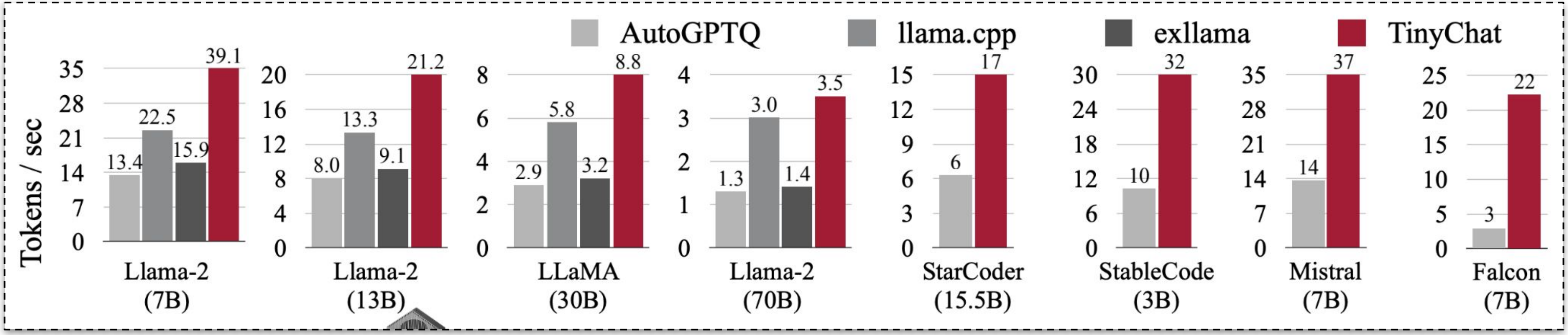
Evaluation

Tested on TinyChat: lightweight system which employs AWQ

Baseline Algorithms: RTN, GPTQ, and GPTQ-Reorder

Baseline Systems: AutoGPTQ, llama.cpp, exllama

Latency tests carried out on NVIDIA Jetson Orin (64GB):



Evaluation

AWQ shows better perplexity-wise performance (for WikiText-2) for a variety of models (e.g., LLaMA) than comparable systems which leverage quantization.

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**

Evaluation

Another benefit of AWQ: robustness with respect to the calibration dataset

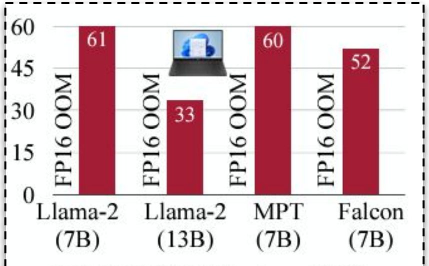
- AWQ has a much smaller perplexity difference than GPTQ when there is a distribution shift from calibration to evaluation
  - Tested with PubMed Abstracts (biomedical) and Enron Emails datasets*

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

* https://arxiv.org/pdf/2101.00027

Weaknesses & Future Directions

- Evaluation does not consider other systems that address the same problem (i.e. run large models w/ limited memory).



(c) RTX 4070 laptop GPU

- Why INT3/4 quantization and not less/more?
  - Could we entirely remove certain weights?

- Other saliency definitions?

LLM in a Flash: Efficient Large Language Model Inference with Limited Memory

Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, S. Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, Mehrdad Farajtabar

Scaling LLMs with Limited Memory

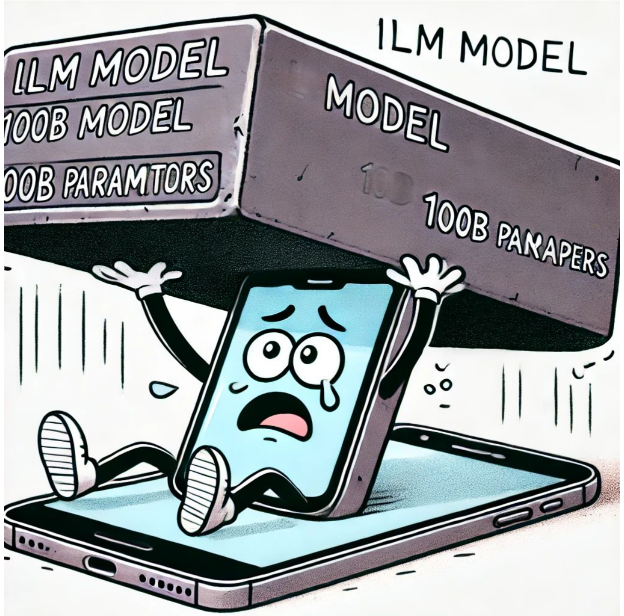
Modern LLMs are MASSIVE!

→ 100 billion ~ trillions of parameters

DRAM is ideal, but limited for **small devices**

→ 7 billion parameters ~ 14GB

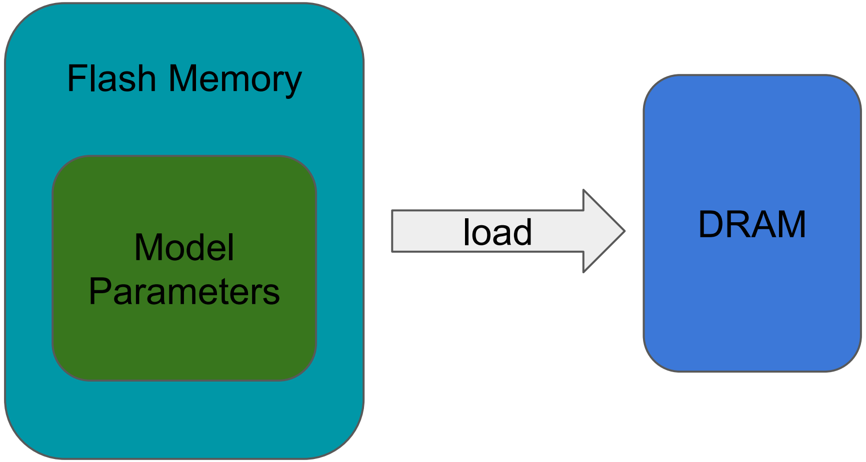
Possible solution? Use **Flash Memory**!



Suggested Approach

Store model parameters in Flash Memory

Load parameters during inference



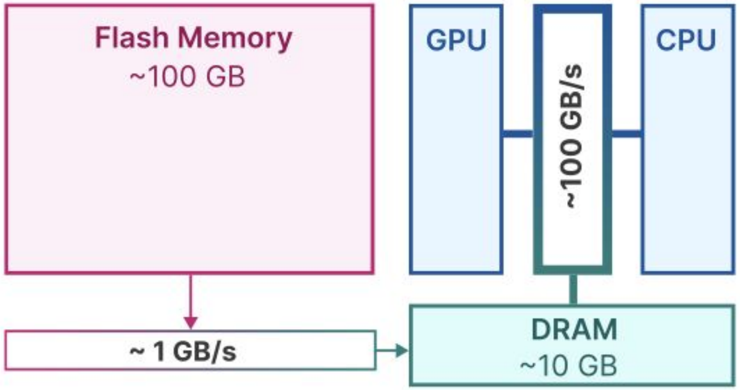
Constraints

Bandwidth/Energy

- DRAM >>> NAND Flash
- Reload entire model for each forward pass

Load Time

- Significant delay for generating first token
- Leverage activation sparsity (covered later)



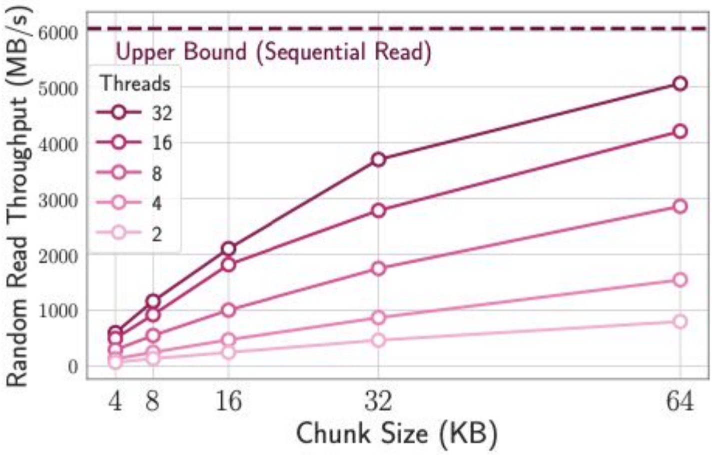
Constraints

Read Throughput

- Flash Memory performs optimally with large sequential reads
- Mac M1 shows 6 GB/s for 1 GB linear read
- Not optimal for random reads

Suggested Solutions

- Read larger chunks of data
- Utilize parallelized reads



(b) Random read throughput of flash memory

Goals

1. Reduce data transfer
2. Increase transfer throughput
3. Efficiently manage loaded data

Feed Forward Network

1. Up Projection

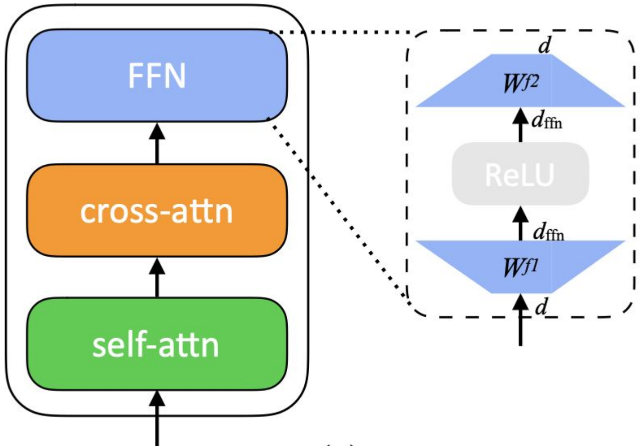
→ Expand dimensions

2. ReLU

→ Induces sparsity

3. Down Projection

→ Reduce dimensions



Reducing Data Transfer

Leverage Activation Sparsity Found in FFN layers

- 97% sparsity for OPT 6.7B model
- 95% sparsity for Falcon 7B model
- 90% sparsity for Llama 2 model

Transfer only essential subset of weights to DRAM

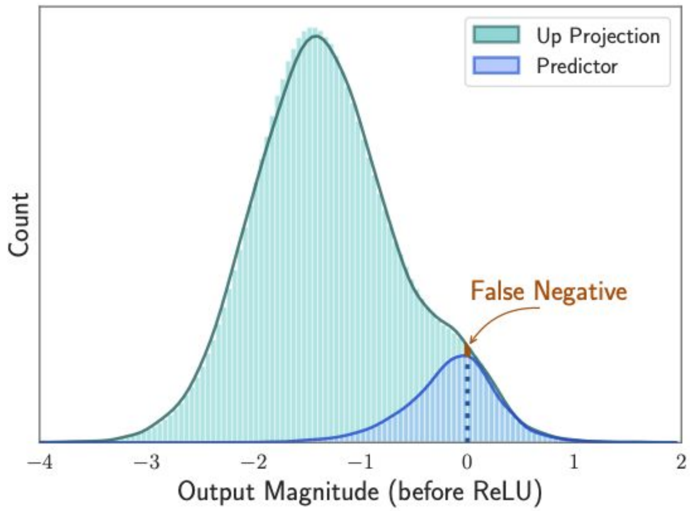
Reducing Data Transfer

Selective Persistence Strategy

- Keep attention weights on DRAM
- Only load active neuron data to DRAM

Anticipating ReLU Sparsity

- ReLU induces more than 90% sparsity
- Low-rank predictor predicts what is zeroed out

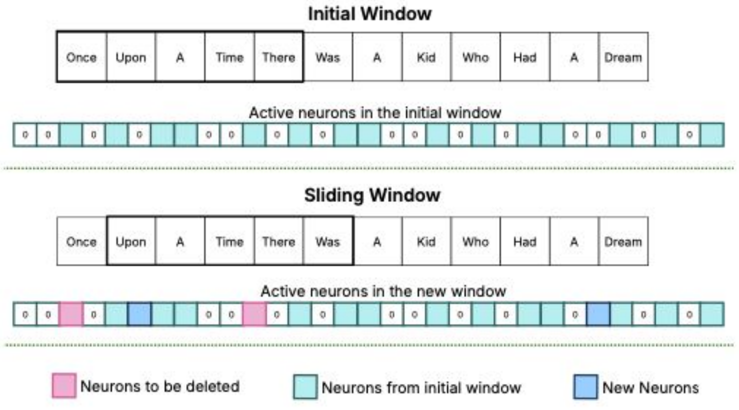


(a) predictor vs relu

Reducing Data Transfer

Sliding Window Technique

- Evict the neurons outside of the window
- Incrementally load neuron data

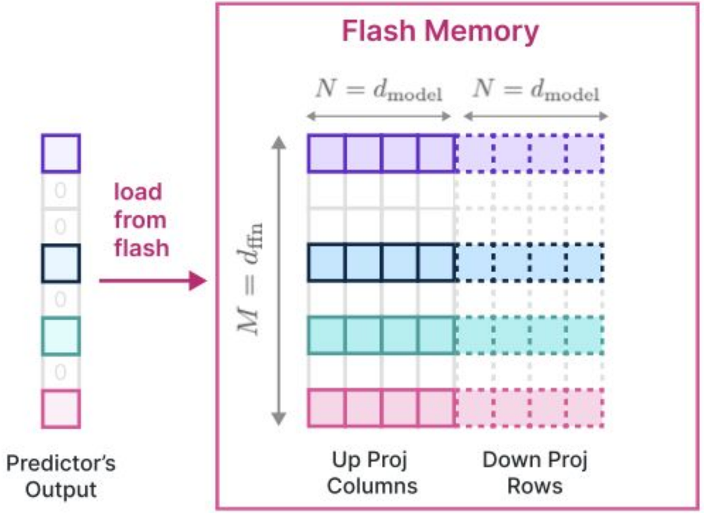


(b) sliding window

Increasing Transfer Throughput

Bundle columns and rows

- Read data in larger chunks
- Number of loads decreases by 1/2

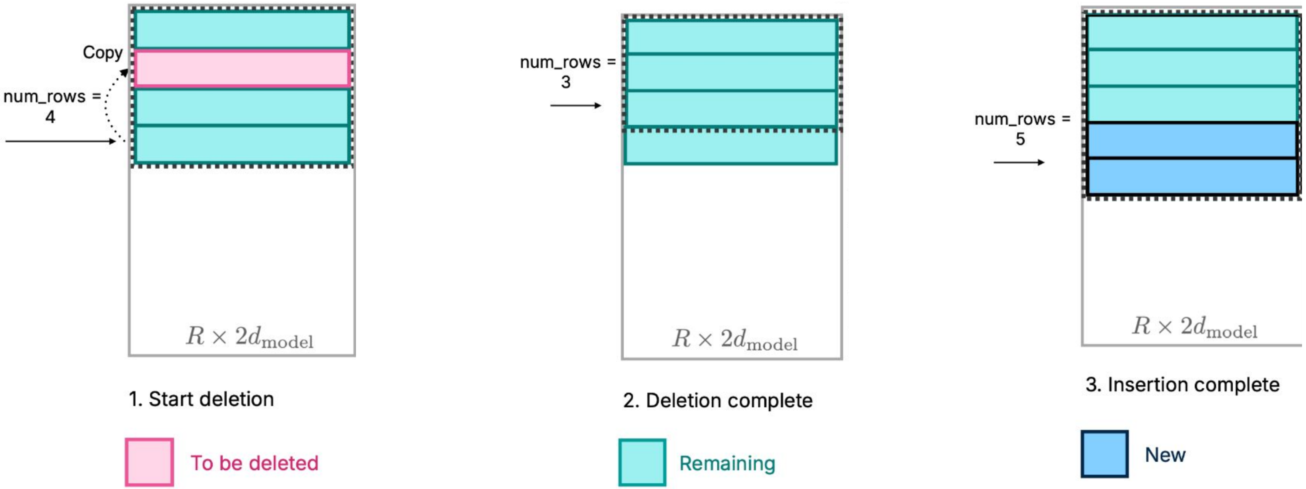


Optimized Data Management in DRAM

Reallocation of memory for new neurons causes latency

→ Preallocate large enough memory for each layer

Prevent memory fragmentation



Evaluation

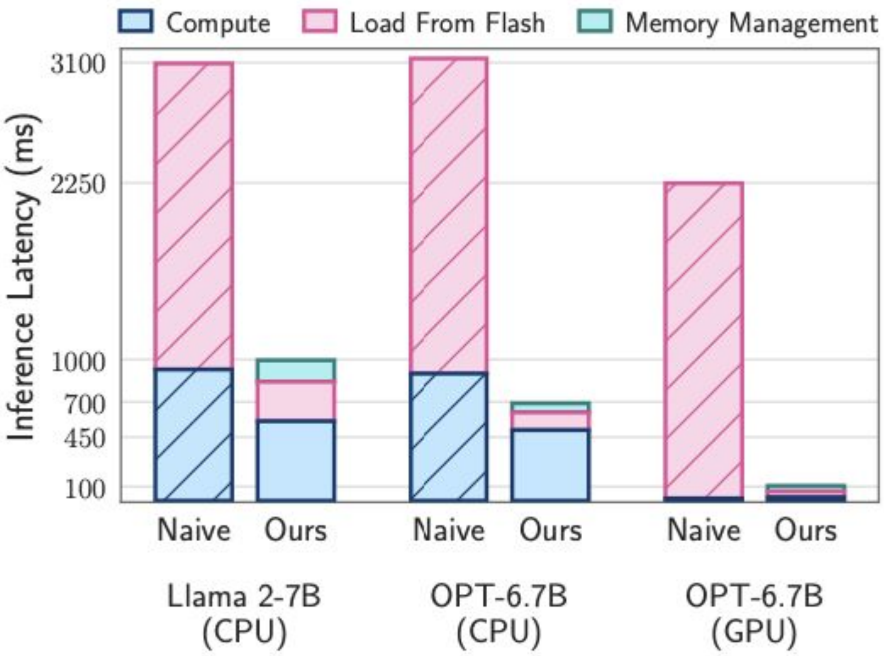
Multiple small reads → decrease throughput

Less data transfer → decrease I/O latency

Bundling → increase throughput

Predictor	Windowing	Bundling	Throughput (GB/s)	I/O Latency (ms)
✗	✗	✗	6.10 GB/s	2196 ms
✓	✗	✗	1.25 GB/s	738 ms
✓	✓	✗	1.25 GB/s	164 ms
✓	✓	✓	2.25 GB/s	87 ms

Evaluation



Compute
Load From Flash
Memory Management
Inference Latency (ms)
3100
2250
1000
700
450
100
Naive
Ours
Naive
Ours
Naive
Ours
Llama 2-7B
(CPU)
OPT-6.7B
(CPU)
OPT-6.7B
(GPU)

Weaknesses & Future Directions

Only single batch inference right now

Assumes 50% model size available in DRAM

Algorithmic Improvements

Thank you for listening!

Definitions and Intuition

Quantization: a technique which reduces the precision of numerical values in a model. Typically used to reduce model size by quantizing weights (e.g. [FP16] → [INT4]).

- Trade model accuracy for decreased memory usage and computation time.

Salient Weights: weights which most influence the model's predictions.

- Typically only ~0.1% - 1.0% of total weights are “salient.”
- Consider some profiler for saliency classification.

Definitions and Intuition

Quantization: If we reduce the precision of the weights, we reduce the total size of the model.

- GPT-3 @ 175B parameters:
    - ~ 87.5 GB in [INT4].
    - ~ 65.6 GB in [INT3].

The model loses accuracy as quantization becomes more aggressive, due to increasing weight imprecision (↑ perplexity).

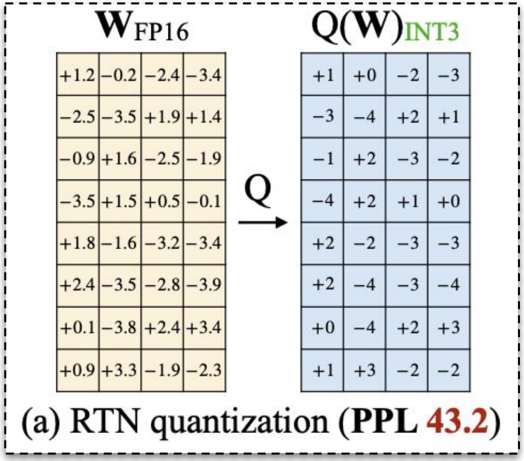


Figure 1:
[FP16] → [INT3] RTN quantization.

AWQ Algorithm

Generally, consider scaling by a factor of ***s*:**

$$\mathrm{Round}(w) \qquad \text{vs} \qquad \frac{1}{s}\mathrm{Round}(s \cdot w)$$

Assuming that the rounding error is similar, the latter has higher output granularity and gives a better estimate.

In fact, assuming equal rounding error, the latter is smaller by the factor of 1/***s***.

TinyChat

Efficient and lightweight embedded system to deploy LLMs on *edge devices*.

- AWQ is an implementational detail of TinyChat.
- Leverages AWQ and custom GPU drivers to deploy LLMs on low-power and low-compute platforms (such as smartphones).

AWQ Algorithm

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

Error for approximating ***w*** by Q(***w***) is $\Delta \cdot \text{RoundError}(\frac{w}{\Delta})$

Idea for Improvement: Scale w before rounding by s and unscale later

Q: Why does this help?

https://www.dropbox.com/scl/fi/dtnp6h6y1mnp7g036axu6/AWQ-slide.pdf?rlkey=ffgh50hxhx8dmsnjiu8kef0ou&e=1&dl=0

AWQ Algorithm

Table illustrating the values of **Δ' / Δ**, error reduction factor, and perplexity for different ***s***, where we scale 1% salient channels:

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 $\Delta^{'}/\Delta$	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