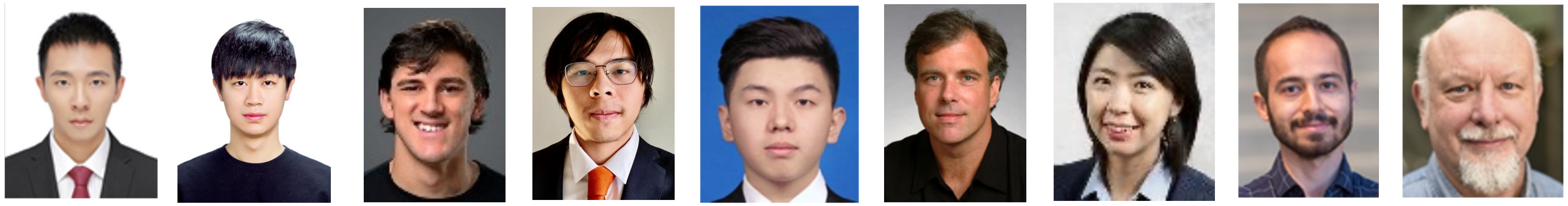
Efficient Inference and Training of Large Neural Network Models

Zhen Dong, Sehoon Kim, Coleman Hooper, Yang Zhou, Sheng Shen, Trevor Darrell, Sophia Shao, Amir Gholami, Kurt Keutzer

University of California at Berkeley

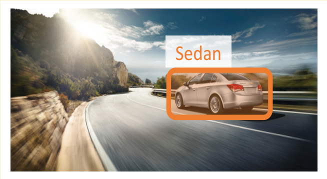


Outline

- Introduction
- SqueezeLLM: Dense-and-Sparse Transformer Quantization
- DQRM: Deep Quantized Recommendation Models
- Conclusion

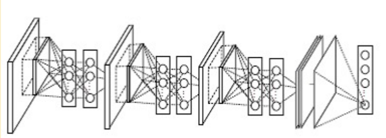
Diverse Application Areas are Converging on one/few DNN Models

Image Classification

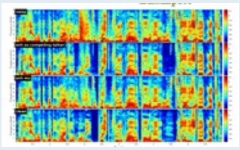


Object Detection

Image Segmentation



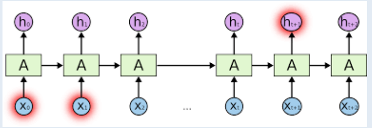
Convolutional NN



Audio Enhancement

Call-center Sentiment Analysis

Speech Recognition

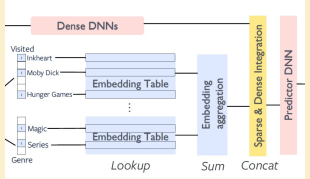


Recurrent NN

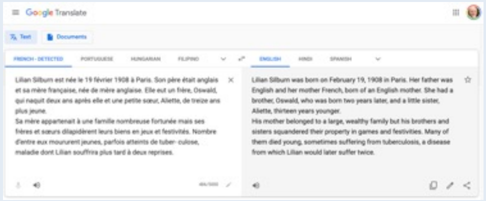
Sentiment Analysis

Music Recommendation

Ad Recommendation



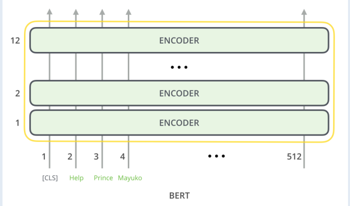
DLRM



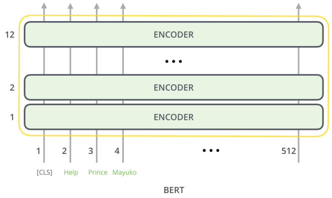
Translation

Question answering

Document Understanding



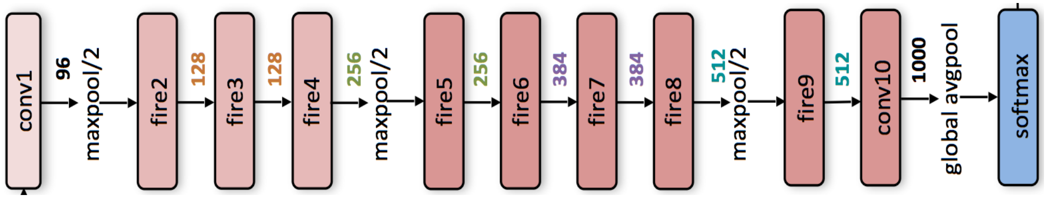
Transformer



Transformer

Our Research Group’s Focus
Efficient Deep Learning/Efficient DNNs

Efficient Inference at the Edge



- **Computer Vision**
  – SqueezeNet, SqueezeNext
  – Shift
  – SqueezeDet
  – SqueezeSeg
- **ASR and NLU**
  – SqueezeWave, SqueezeBERT
  – SqueezeFormer
  – SqueezeLLM

Squeeze Family of DNNs

Efficient/Scalable Training and Inference in the Cloud

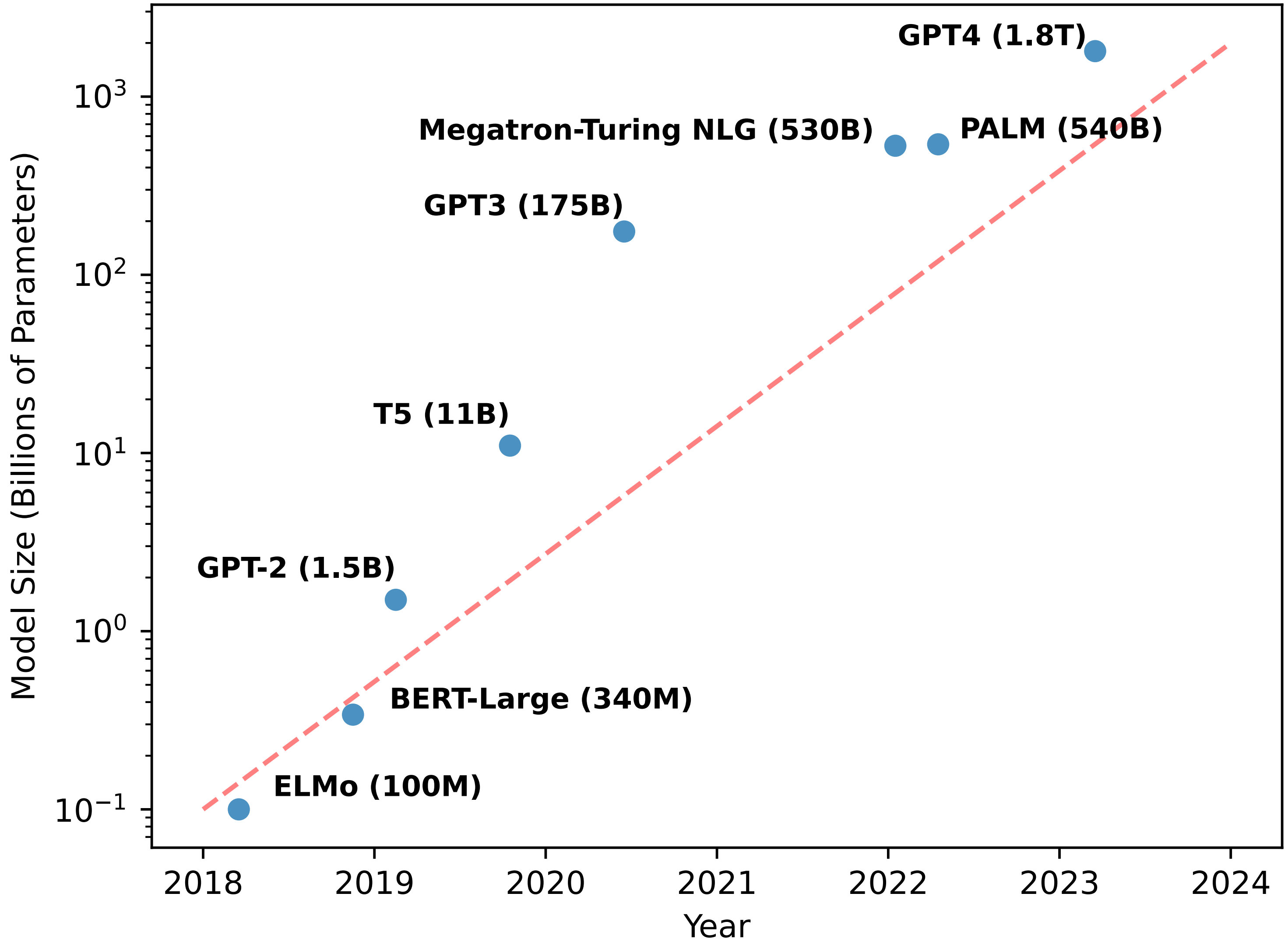
Efficient Training
- FireCaffe, LARS, LAMB
- Staged-Training

Efficient Inference
- HAWQ, HAWQV2, HAWQV3
- Learned Token Pruning

Invited/Keynote Speaker at EMDNN (NeurIPS 2016), ESWEEK 2017, EDLCV (CVPR 2017), CVPRAD (CVPR 2018) MLPCD NeurIPS (2018) LPIRC (2019), EMC^2 (NeurIPS 2019), HENP (ESWEEK 2020), EVW (ICLR 2021), ENLP (2021), Design Automation Conference 2021, VLSI SOC 2022, MLSYSArc (ISCA 2022), SustaiNLP (EMNLP 2022)

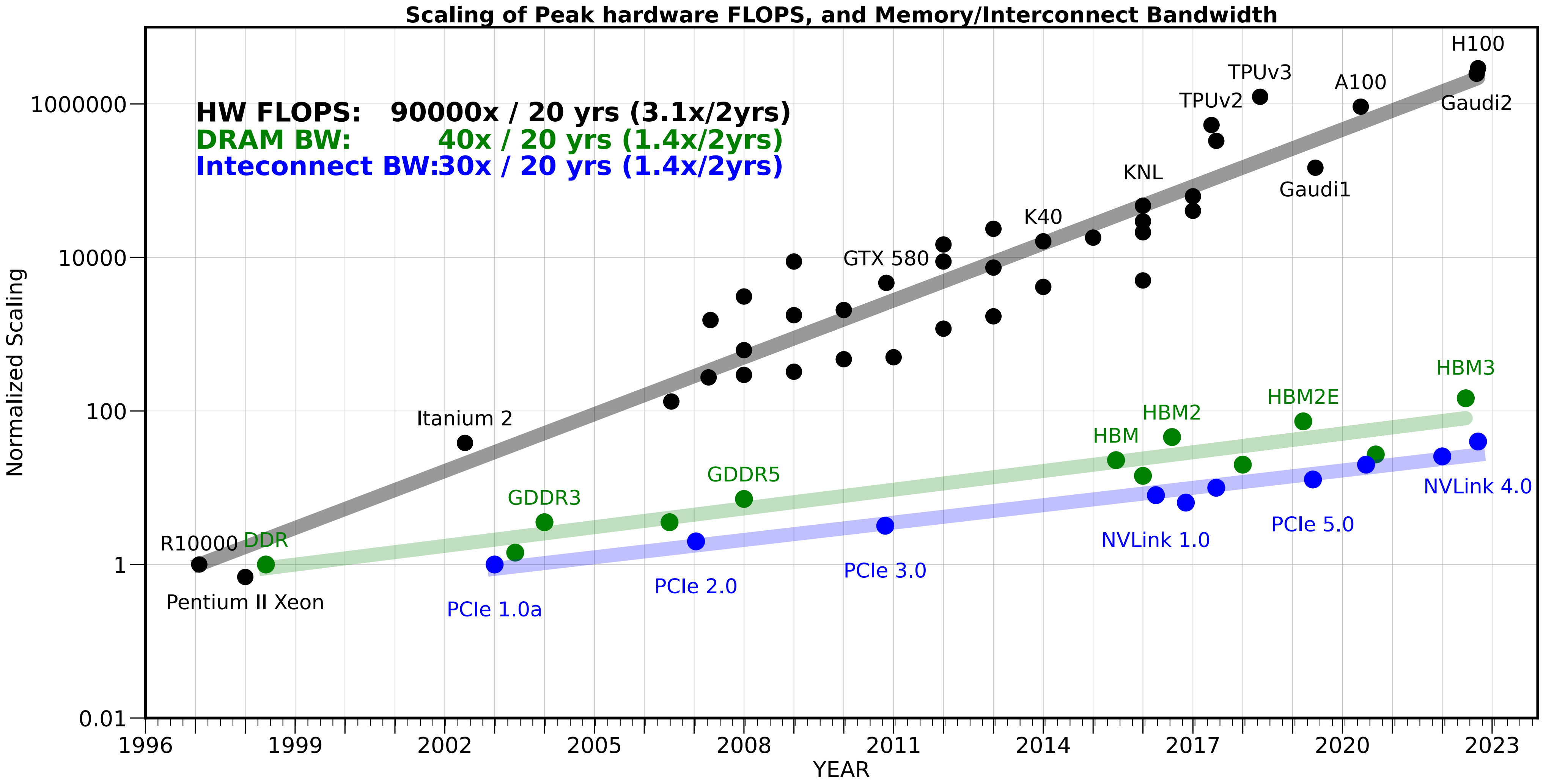
Growth of LLMs

18000x growth in model size over the past 5 years



Adapted from: https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/.

Memory Wall: Main Bottleneck is Memory Bandwidth



Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W. Mahoney, Kurt Keutzer, [AI and Memory Wall], Riselab Medium Blogpost, 2021.

Outline

- Introduction
- SqueezeLLM: Dense-and-Sparse Transformer Quantization
- DQRM: Deep Quantized Recommendation Models
- Conclusion

The Field is Changing Super Fast

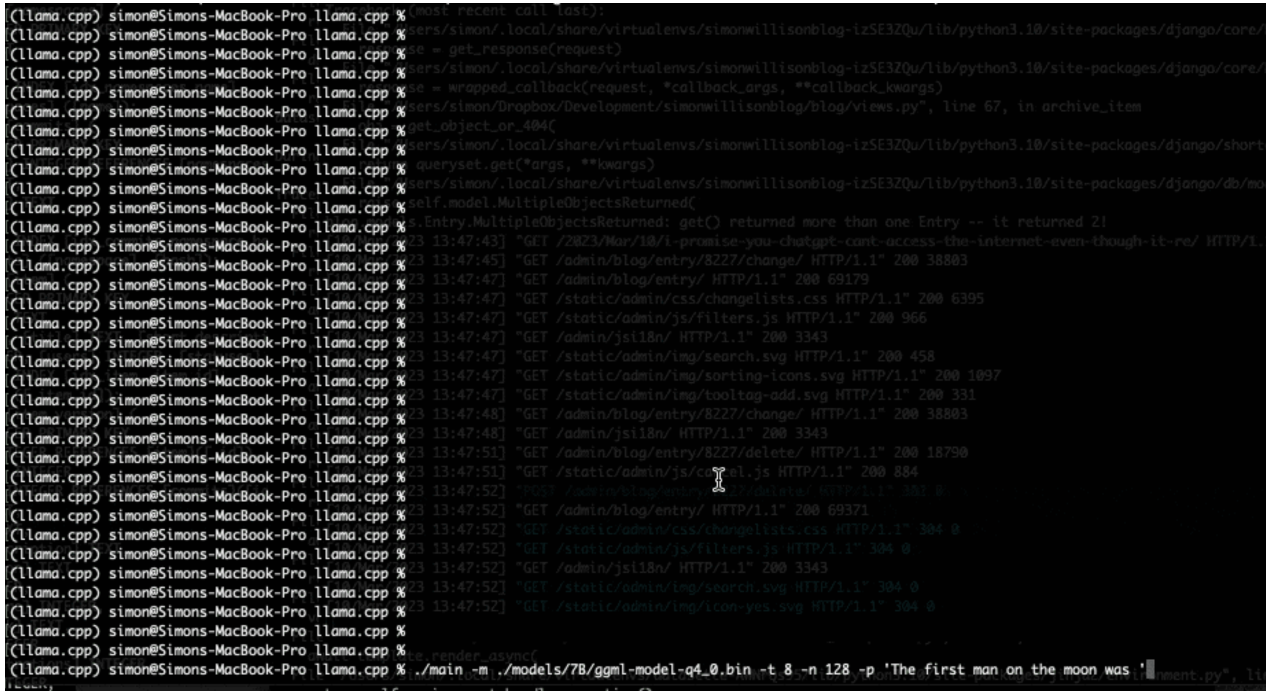
GPT-3

ChatGPT

Stanford Alpaca

Promising Preliminary Results

LLaMA.cpp
- Ports LLaMA models in C/C++ and allows running them on personal computers
- Over 32.5k stars on github, active community

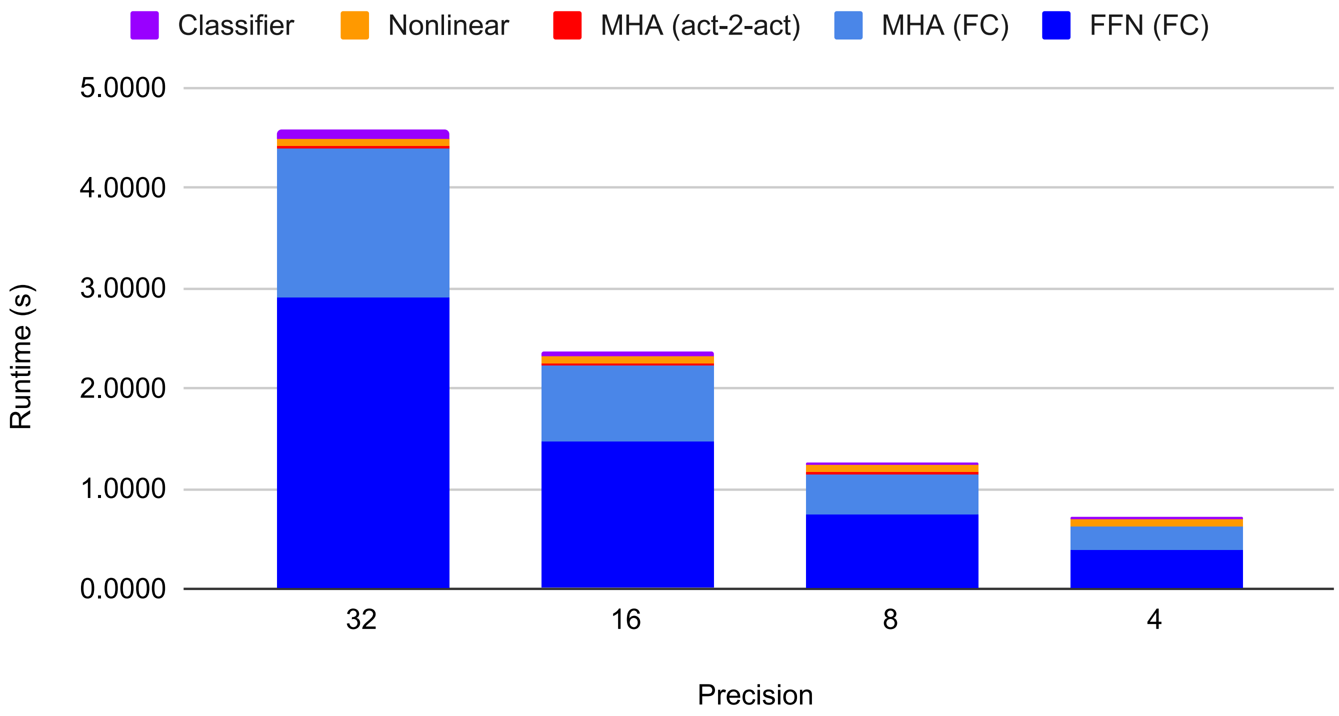


Running LLaMA 7B on 65GB M2 Macbook Pro

Memory Wall for Inference!

The dominant contributor to runtime is the time for **memory bandwidth not compute**

Breakdown of LLaMA 7B model with Seq Len of 128 and batch of 1



S. Kim*, C. Hooper*, A. Gholami*, Z. Dong, X. Li, S. Sheng, M. Mahoney, K. Keutzer, SqueezeLLM: Dense-and-Sparse Quantization, arxiv

Quantization is Critical for Efficient and Fast LLMs

- Quantization reduces the **memory footprint** and **peak memory** requirement
  - 3bit LLaMA-7B: 13GB → 3GB
- It will also improve the **inference latency**

- However, achieving **performance** with low-bit precision remains **challenging**
  - Particularly with very **low bit precision** (e.g. 4bit and lower)
  - And for relatively **smaller models** (e.g. < 50B parameters)

How We Reduced the Memory Requirement

- **Sensitivity-Aware Non-uniform Quantization**
  - Lookup table (LUT) based non-uniform quantization

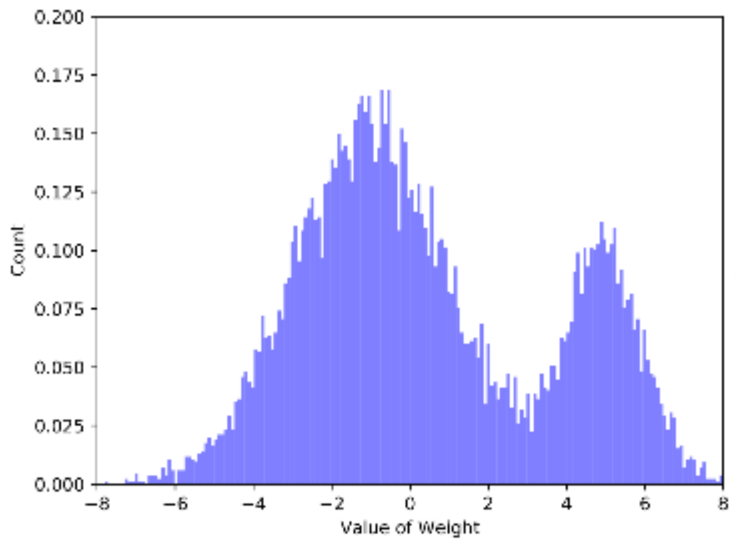
- **Dense-and-Sparse Quantization**
  - Decompose a matrix into a dense matrix and a sparse matrix

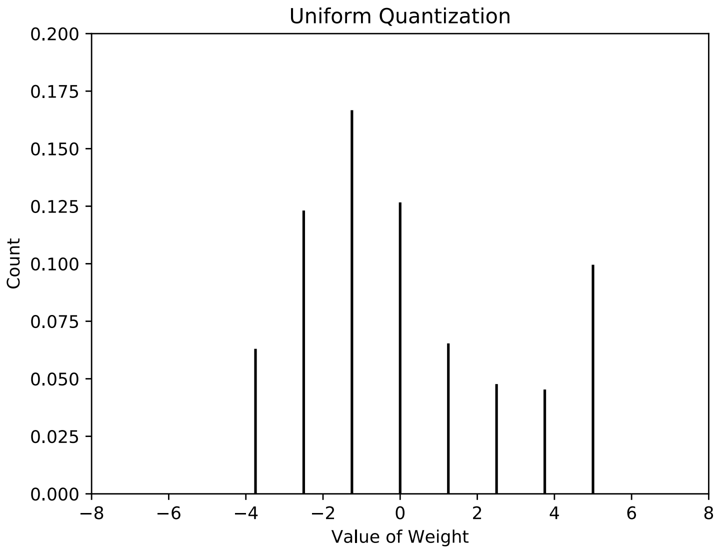
Uniform vs Non-Uniform Quantization

Uniform Weight-only Quantization: Sub-optimal for 2 reasons

1. Weight distribution in neural networks is typically **nonuniform**
2. The main bottleneck is **memory and not compute**

$$Q(w)^* = \underset{Q}{\arg\min} \|W - W_Q\|_2^2$$





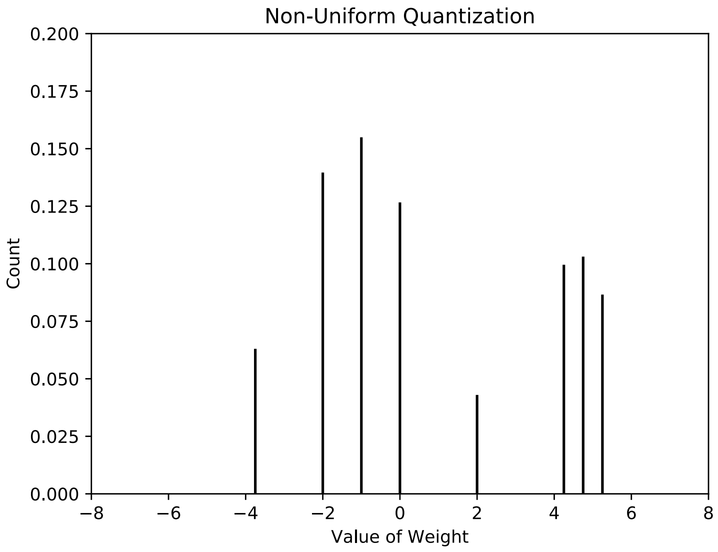
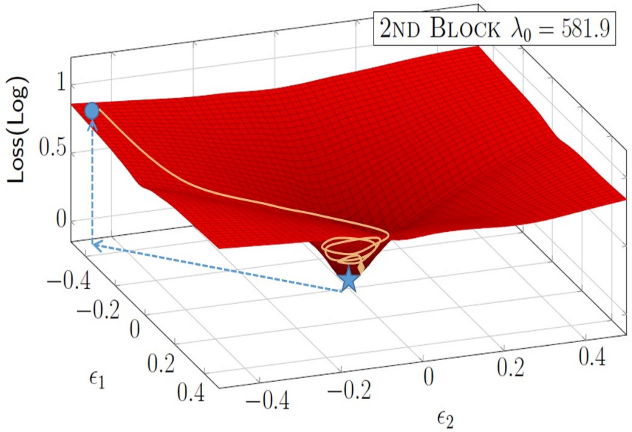


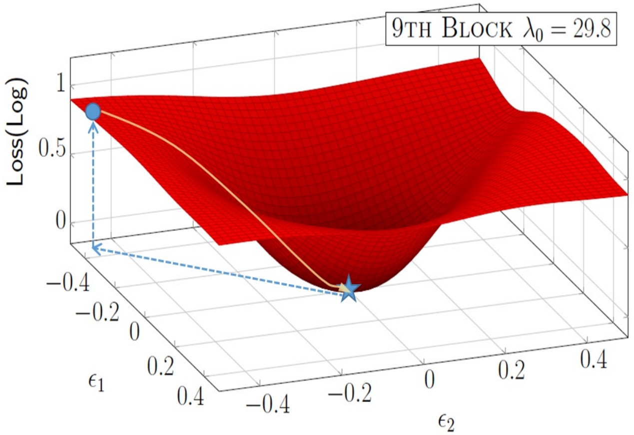
Figure from: Park, Eunhyeok, Sungjoo Yoo, and Peter Vajda. "Value-aware quantization for training and inference of neural networks." ECCV, 2018.

Non-Uniform Quantization

Issue: Some weights are more sensitive than others with respect to quantization errors



Sharper Loss Landscape
More sensitive to quantization error



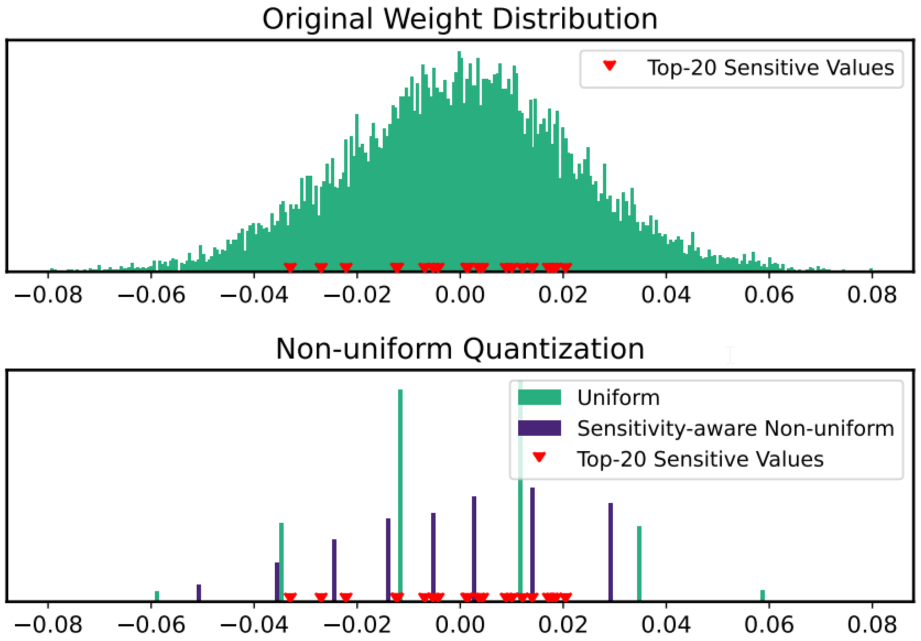
Flatter Loss Landscape
Less sensitive to quantization error

Z. Yao*, Z. Dong*, Z. Zheng*, A. Gholami*, E. Tan, J. Li, L. Yuan, Q. Huang, Y. Wang, M. W. Mahoney, K. Keutzer, HAWQ-V3: Dyadic Neural Network Quantization in Mixed Precision, ICML, 2021.
Z. Dong, Z. Yao, D. Arfeen, A. Gholami, M. Mahoney, K. Keutzer, HAWQ-V2: Trace Weighted Hessian Aware Quantization, NeurIPS 2020.

Sensitive-aware Non-uniform Quantization

Better Problem Definition: $Q(w)^* = \underset{Q}{\arg\min} \|W - W_Q\|_2^2 \Rightarrow \underset{Q}{\arg\min} \sum_{i=1}^{N} \boxed{\mathcal{F}_{ii}} (w_i - Q(w_i))^2.$

Sensitivity metric: Fisher diagonal



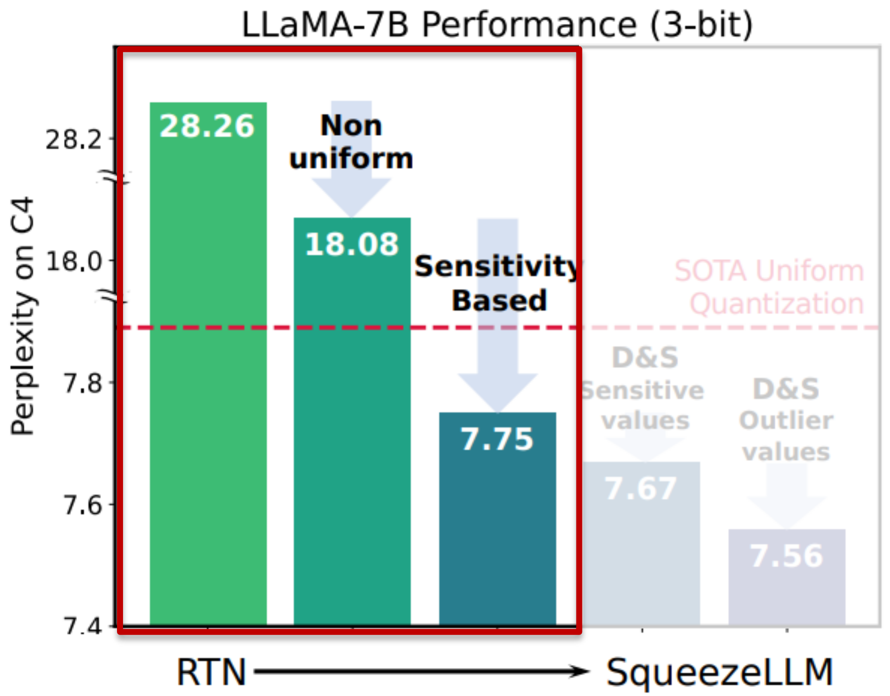
Instead of treating all the weights same, scale them with the Hessian values allocate quantization centroids near more sensitive values

Sensitive-aware Non-uniform Quantization

Better Problem Definition: $$Q(w)^* = \arg\min_{Q} \|W - W_Q\|_2^2 \Rightarrow \arg\min_{Q} \sum_{i=1}^{N} \boxed{\mathcal{F}_{ii}}(w_i - Q(w_i))^2.$$

Sensitivity metric: Fisher diagonal

~0.2 PPL Better than SOTA



How We Reduced the Memory Requirement

❖ **Sensitivity-Aware Non-uniform Quantization**

- Lookup table (LUT) based non-uniform quantization

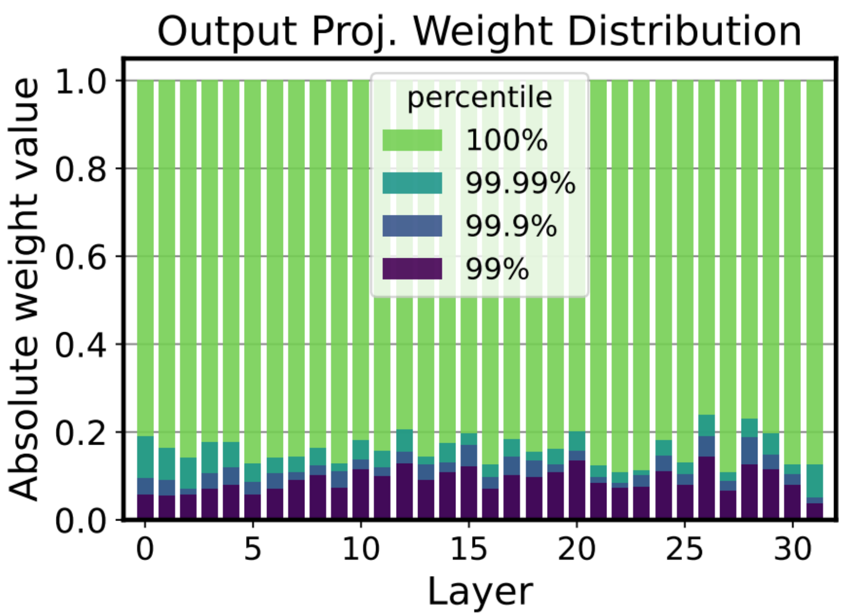
❖ **Dense-and-Sparse Quantization**

- Decompose a matrix into a dense matrix and a sparse matrix

Outliers

- **Weight distribution analysis of LLaMA-7B Model**
  - **Range of the weight values** in the Output (MHA) and Down (FFN) projection layers
  - Around **99.99%** of the values are in the **10-20%** of the overall range

- **Outliers over-exaggerate the quantization range**
  - Grouping to circumvent the issue: outliers in one group would not affect other groups
  - This is not a **direct solution**, and can be **costly** with non-uniform quantization



Dense-and-Sparse Decomposition

- Decompose a matrix into a **dense matrix** and a **sparse matrix**

$$W = \boxed{D} + \boxed{S}$$

Dense matrix: reduced range
→ smaller quantization error

Sparse matrix: ~0.1% outliers

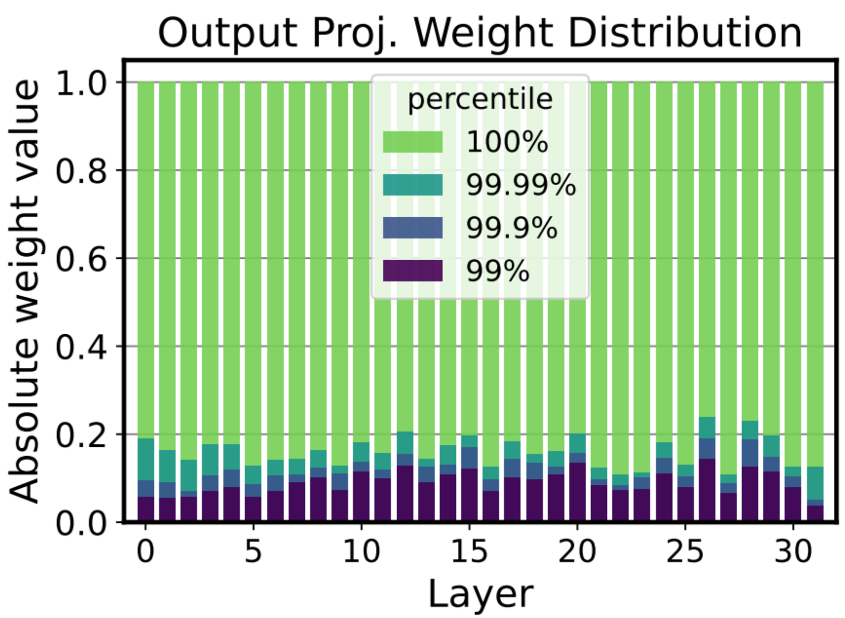
$$A = \begin{pmatrix} 7.5 & 2.9 & 2.8 & 2.7 & 0 & 0 \\ 6.8 & 5.7 & 3.8 & 0 & 0 & 0 \\ 2.4 & 6.2 & 3.2 & 0 & 0 & 0 \\ 9.7 & 0 & 0 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5.8 & 5.0 \\ 0 & 0 & 0 & 0 & 6.6 & 8.1 \end{pmatrix}$$

rowptr: (0 4 7 10 12 14 16)

colind: (0 1 2 3 0 1 2 0 1 2 0 3 4 5 4 5)

val: (7.5 2.9 2.8 2.7 6.8 5.7 3.8 2.4 6.2 3.2 9.7 2.3 5.8 5.0 6.6 8.1)

Sparse matrix representation using the compressed row storage (CSR) format



Dense-and-Sparse Decomposition

- Decompose a matrix into a **dense matrix** and a **sparse matrix**

$$Wx = (\boxed{D} + \boxed{S})x = Dx + Sx \approx \boxed{Qx} + \boxed{Sx}$$

Dense matrix: reduced range
→ smaller quantization error

Sparse matrix: ~0.1% outliers

Sparse matrix multiplication
(e.g. CuSparse)

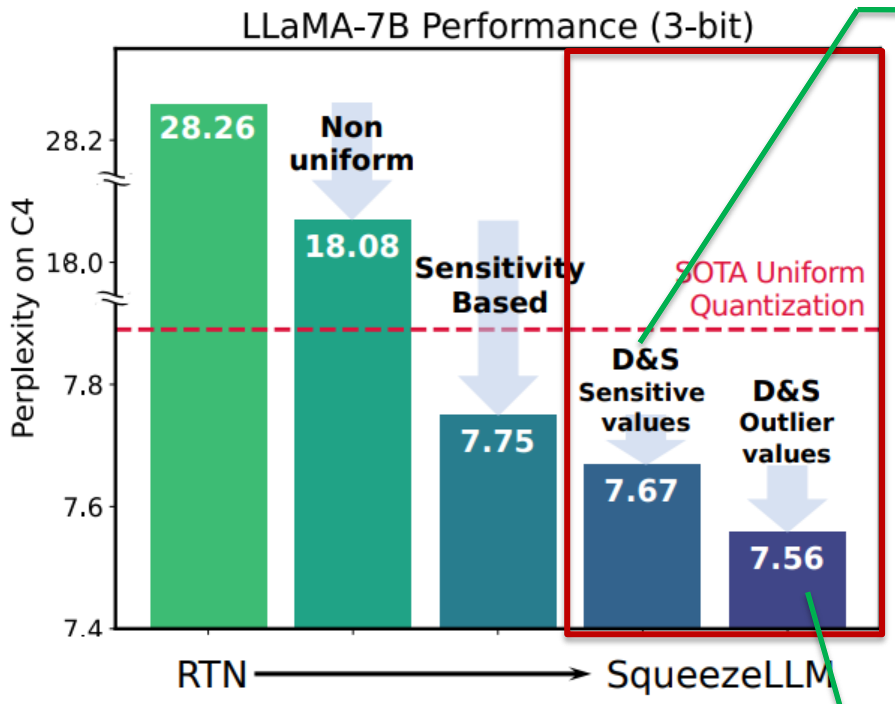
FP16 **dense matrix multiplication**
After dequantization

$$A = \begin{pmatrix} 7.5 & 2.9 & 2.8 & 2.7 & 0 & 0 \\ 6.8 & 5.7 & 3.8 & 0 & 0 & 0 \\ 2.4 & 6.2 & 3.2 & 0 & 0 & 0 \\ 9.7 & 0 & 0 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5.8 & 5.0 \\ 0 & 0 & 0 & 0 & 6.6 & 8.1 \end{pmatrix}$$

```
rowptr:  ( 0   4   7   10  12  14  16 )
colind:  ( 0   1   2   3   0   1   2   0   1   2   0   3   4   5   4   5 )
val:     ( 7.5 2.9 2.8 2.7 6.8 5.7 3.8 2.4 6.2 3.2 9.7 2.3 5.8 5.0 6.6 8.1 )
```

Sparse matrix representation using the compressed row storage (CSR) format

Dense-and-Sparse Decomposition

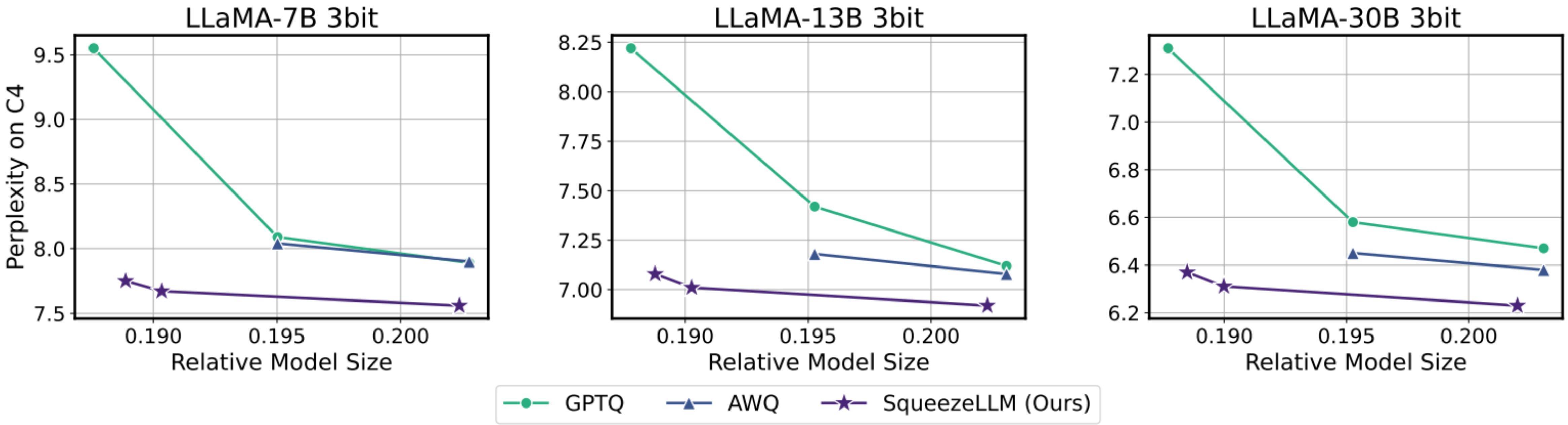


We can also include **"sensitive" values** into the sparse matrix (**0.05%**) so that they are preserved in the **FP16 format**

Then, we include **outlier values (0.4%)** to restrict the quantization range

Additional 0.2 PPL improvement
With 3-bit quantization (5x compression), <0.5 PPL off from FP16

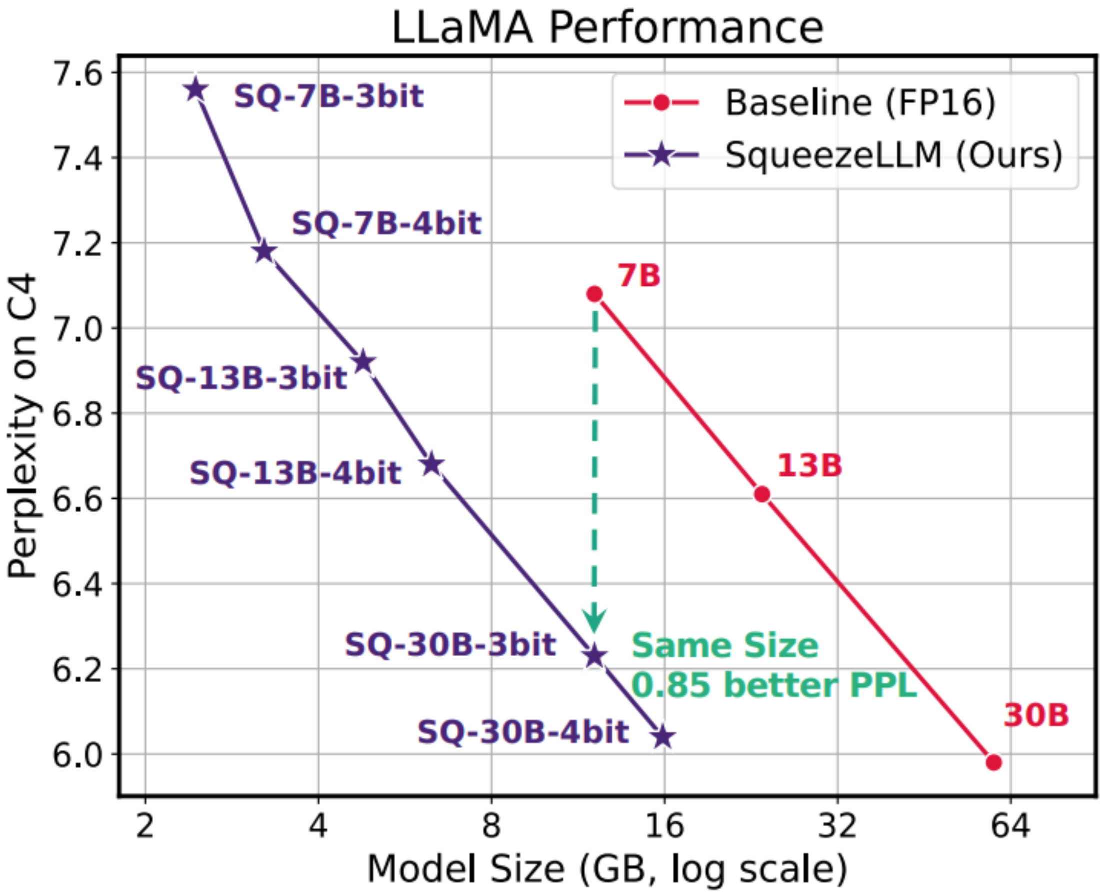
SqueezeLLM vs Other Quantization Methods



With the same model size, our method **always outperforms GPT-Q and AWQ**

Frantar, Elias, et al. "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers." *arXiv preprint arXiv:2210.17323* (2022).
Lin, Ji, et al. "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration." *arXiv preprint arXiv:2306.00978* (2023).

Overall Results:
Better Performance for Similar Memory Footprint!



Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., Shen, S., Mahoney, M.W. and Keutzer, K. SqueezeLLM: Dense-and-Sparse Quantization. *arXiv:2306.07629*.

Quantization of Instruction Following Models

Quantize the instruction-tuned **Vicuna-7B and 13B** models.

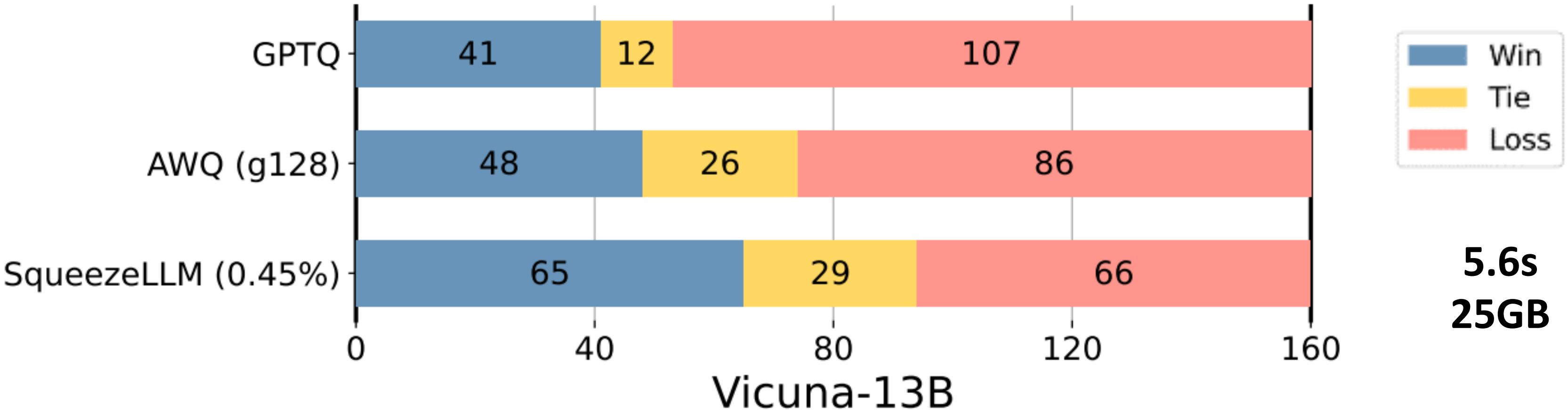
Zero-shot MMLU Evaluation: Testing domain-specific knowledge and problem solving ability

Model	Peak Memory	Latency	Accuracy
Vicuna-7B	14GB	3.2s	39.1
SQ-Vicuna-13B-4bit	6.9GB	**2.8s**	**39.2**
SQ-Vicuna-13B-3bit-0.45%	**5.9GB**	3.4s	**39.4**
SQ-Vicuna-13B-4bit-0.45%	**7.4GB**	3.6s	**41.0**

Quantization of Instruction Following Models

GPT-4 based evaluation of instruction-following ability after quantization

3.4s
5.9GB



5.6s
25GB

Win/Tie/Loss when comparing text generated quality of the quantized models (left) vs. the FP16 baseline (right)
-> Quantized models shows similar performance to the FP16 baseline despite being significantly smaller

Intel AI tools and frameworks

- **How we used Intel's tools/frameworks**
  - Intel's vLab played a key role in developing SqueezeLLM, especially for speeding up the sensitivity-based non-uniform quantization algorithm through parallelization.

- **Integrating SqueezeLLM with oneAPI for its flexible cross-platform capability.**
  - From our initial implementation, we are observing its potential on both GPUs and CPUs.
  - Enhancing the cross-platform support is the next step in our roadmap for extending SqueezeLLM's accessibility.

Outline

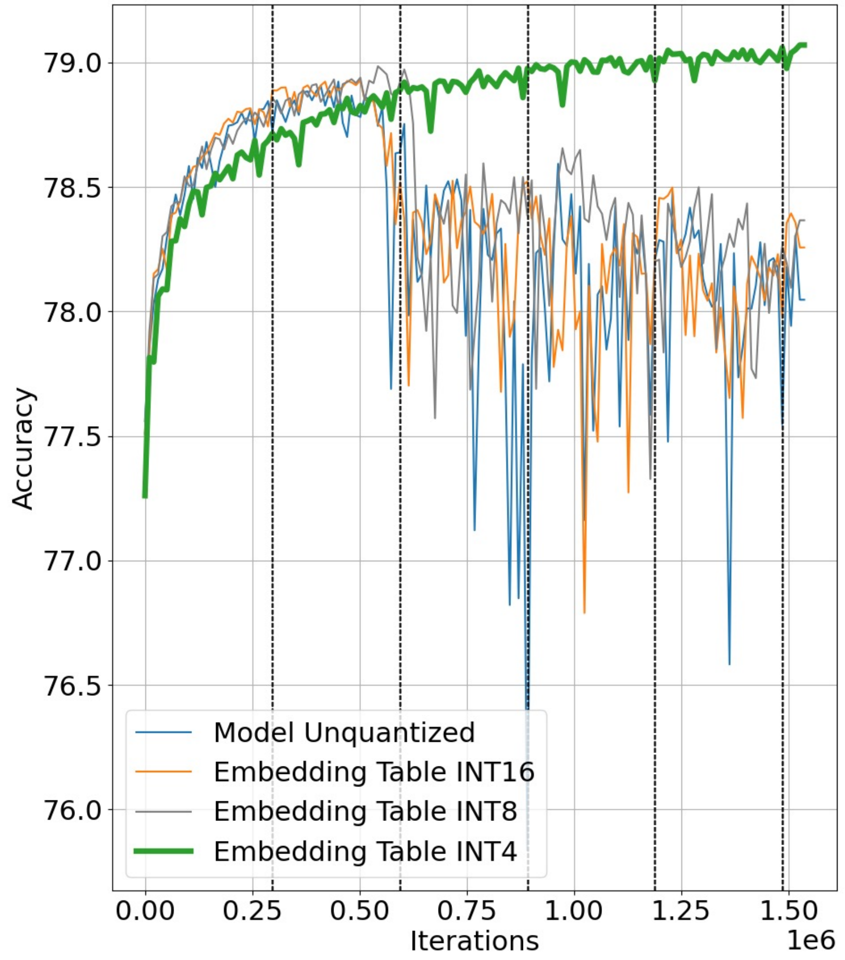
- Introduction
- SqueezeLLM: Dense-and-Sparse Transformer Quantization
- DQRM: Deep Quantized Recommendation Models
- Conclusion

Weights Quantization in INT4 reduces overfitting

- Accuracy obtained from multi-GPUs is slightly **worse than** on single GPUs in all settings (Kaggle dataset), but trend on the single-node is is **consistent** with that on multi-node
- To better demonstrate weight quantization performance, experiment results using a single GPU are presented.

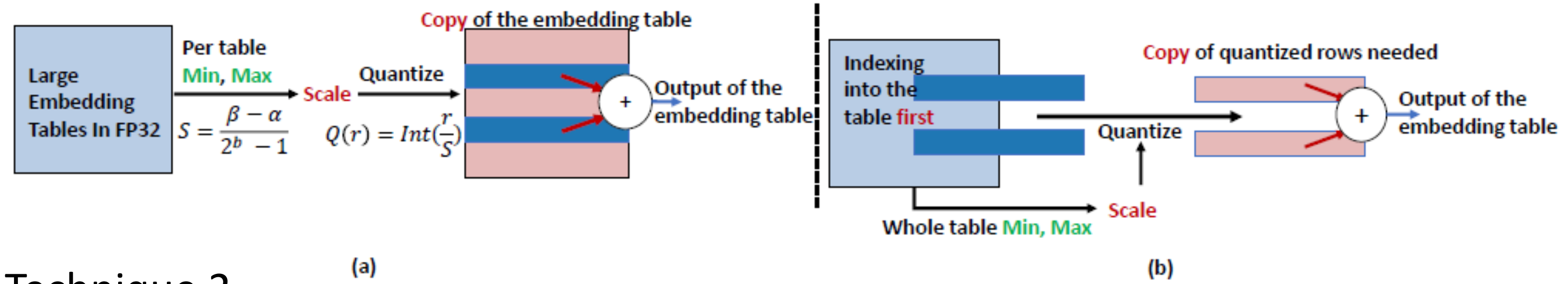
Table 2: DLRM Embedding Tables Quantization, accuracies evaluated on the Kaggle Dataset

Quantization Bit Width	Testing Accuracy	Testing ROC AUC
Full-precision	78.923%	0.8047
INT16	78.928% (+0.005%)	0.8046 (-0.0001)
INT8	78.985% (+0.062%)	0.8054 (+0.0007)
INT4	**79.070% (+0.147%)**	**0.8075 (+0.0028)**



Weights Quantization Results from single node

Technique 1.



Technique 2.

- Vertical axis is the **wall-clock time contribution**
- Both CPU and GPU time are normalized (don't compare between two columns)
- Finding Scale is expensive, we use **periodic update**

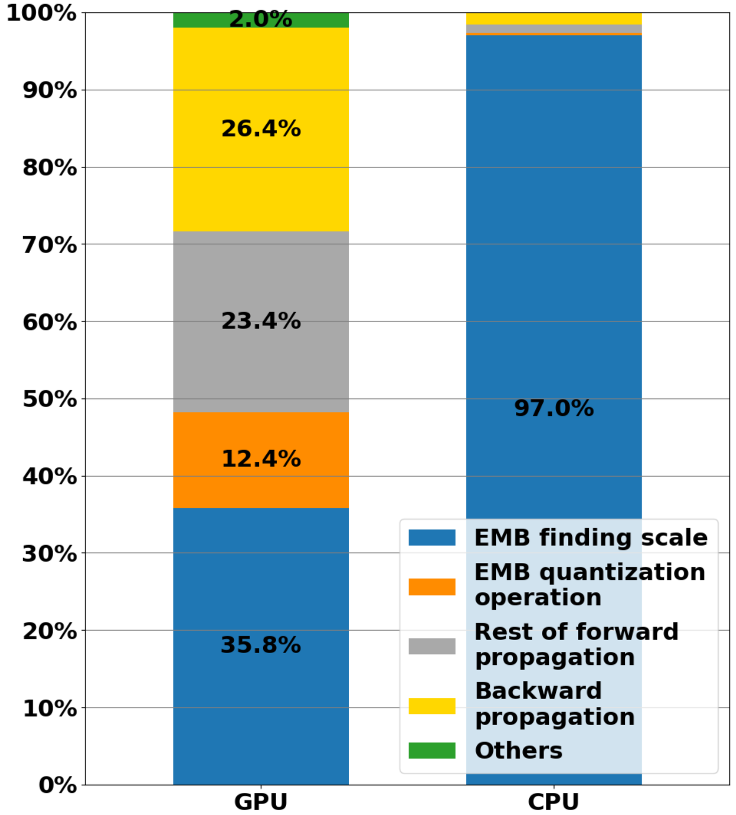
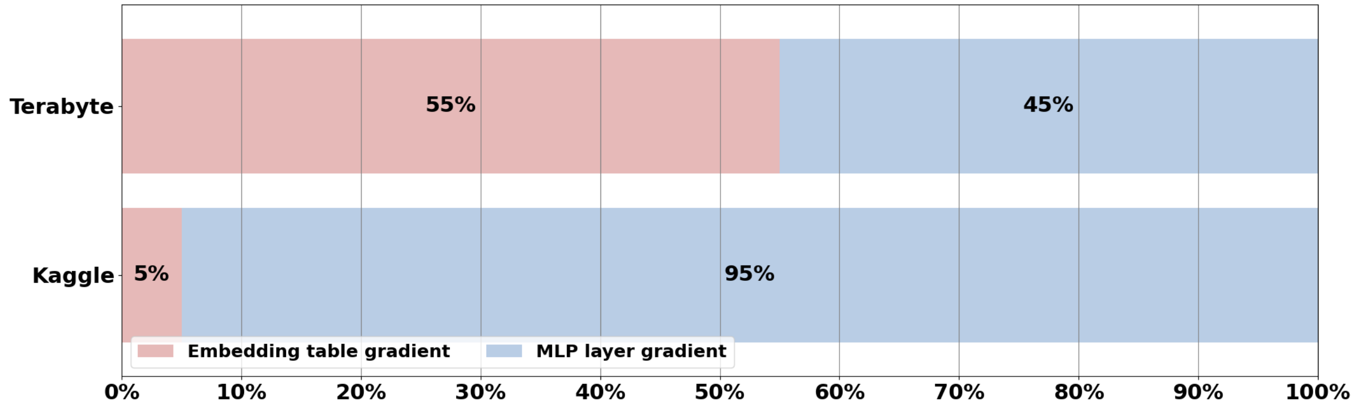


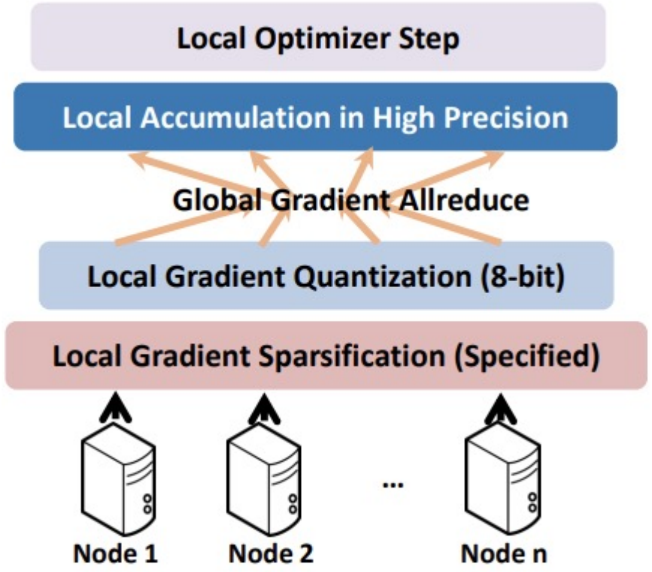
Table 5: Evaluation of Periodic Update on Kaggle and Terabyte Datasets

Model Settings	Period	Latency per iter	Testing Accuracy	ROC AUC
Kaggle	1	31 ms	79.040%	0.8064
	200	22 ms	79.071%	0.8073
	500	22 ms	79.034%	0.8067
Terabyte	1	>1200 ms	-	-
	200	58 ms	81.159%	0.7998
	500	51 ms	81.193%	0.8009
	1000	46 ms	81.210%	0.8015

Why MLP gradient should also be taken into account?

- During gradient communication, a local gradient sparsification process is first added
- I previously underestimate the effect of gradient sparsification
- EMB gradient are drastically reduced after the process, which makes the MLP gradient a component of the gradient communication that cannot be overlooked
- To find out the exact proportion, I have to profile the gradient experimentally, as the rough estimate are always not accurate because of the irregular embedding vector access pattern from power law





Fully quantized gradient communication

- MLP gradients are **sensitive to quantization**
- Utilize **previous error compensation** technique to compensate MLP gradient specifically

Table 4. Communications compression for Distributed Data Parallelism training among four nodes or GPUs

Model Settings	Training Platforms	Communication Compression settings	Communication Overhead per iter	Latency per iter	Training Loss	Testing Accuracy	ROC AUC
Kaggle	4X	gradient uncompressed (DQRM 4-bit)	2.161 GB	>1000 ms	0.436685	78.897%[1]	0.8035
	Nvidia A5000	+ EMB gradient sparsification (specified)[2]	2.010 MB	61 ms	0.436685	78.897%	0.8035
	GPUs	+ INT8 gradient Quantization	0.509 MB	110 ms[3]	0.442300	78.840%	0.8023
Terabyte	2X (2 processes)	gradient uncompressed (DQRM 4-bit)	12.575 GB	>1000 ms	0.412688	81.156%	0.7997
	Intel(R) Xeon(R)	+ EMB gradient sparsification (specified)	6.756 MB	210 ms	0.412688	81.156%	0.7997
	Platinum 8280 CPU	+ INT8 gradient Quantization	1.732 MB	225 ms	0.414731	81.035%	0.7960

[a] data parallelism consistently lowers the test accuracy in all settings compared with single-node training
[b] Specified sparsification is a lossless compression for embedding tables so the testing accuracy is exactly the same as uncompressed case
[c] PyTorch sparse tensor allreduce library doesn't support low-precision arithmetic, without further system level effort in low-precision optimization, the latency per iteration increases purely from the quantization overhead per iteration

Stacking up with Previous QAT Techniques

Previously, reviews want more comparisons.
We implemented PACT, LSQ, HAWQ quantization-aware training techniques on DLRM models

Table 3. DQRM 4-bit quantization results evaluated on Kaggle and Criteo datasets

(a) 4-bit quantization for DLRM on Kaggle

Quant Settings	Model Bit Width	Model Size	Training loss	Training time/it	Testing Accuracy	Testing ROC AUC
Baseline	FP32	2.161 GB	0.304	7 ms	78.923%	0.8047
Vanilla PTQ	INT4	0.270 GB	-	-	76.571%	0.7675
PACT* [2]	INT4	0.270 GB	Cannot Converge			
(MLP in FP32)		0.271 GB	0.303	69 ms	78.858%	0.8040
LSQ [7]	INT4	0.270 GB	0.350	25 ms	78.972%	0.8051
(MLP in FP32)		0.271 GB	0.352	21 ms	78.987%	0.8059
HAWQ [6]	INT4	0.270 GB	0.437	31 ms	79.040%	0.8064
(MLP in FP32)		0.271 GB	0.436	27 ms	79.070%	0.8075
DQRM (Ours)	INT4	0.270 GB	0.437	22 ms	79.071%	0.8073
(MLP in FP32)		0.271 GB	0.436	20 ms	79.092%	0.8073

(b) 4-bit quantization for DLRM on Terabyte

Quant Settings	Model Bit Width	Model Size	Training loss	Training time/it	Testing Accuracy	Testing ROC AUC
Baseline	FP32	12.575 GB	0.347071	19 ms	81.165%	0.8004
Vanilla PTQ	INT4	1.572 GB	-	-	78.681%	0.7283
PACT* [7]	INT4	1.572 GB	Cannot Finish >1000 ms/it			
HAWQ [6]	INT4	1.572 GB	Cannot Finish >1000 ms/it			
LSQ [7]	INT4	1.572 GB	0.350	42 ms	81.134%	0.7996
(MLP in FP32)		1.572 GB	0.356	42 ms	81.127%	0.7998
DQRM (Ours)	INT4	1.572 GB	0.409774	29 ms	81.210%	0.8015
(MLP in FP32)		1.572 GB	0.412	29 ms	81.200%	0.8010

*PACT [2] uses DoReFa [36] for weight quantization

Gradient Sparsification and Quantization

Sparsification: only communicate gradient values that are used and nonzero.

Quantization: Use uniform quantization on gradients

GPUs – only “gloo” backend is available, but it has many restrictions.

CPUs – “gloo”, “mpi”, and **“one_ccl”**, we use **“one_ccl”** for the best support and optimization.

Backend	gloo		mpi		nccl	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✗	✓	?	✗	✓
recv	✓	✗	✓	?	✗	✓
broadcast	✓	✓	✓	?	✗	✓
all_reduce	✓	✓	✓	?	✗	✓
reduce	✓	✗	✓	?	✗	✓
all_gather	✓	✗	✓	?	✗	✓
gather	✓	✗	✓	?	✗	✓
scatter	✓	✗	✓	?	✗	✗
reduce_scatter	✗	✗	✗	✗	✗	✓
all_to_all	✗	✗	✓	?	✗	✓
barrier	✓	✗	✓	?	✗	✓

Summary

We studied efficient inference and training of large neural network models.

- Our proposed dense-and-sparse quantization can effectively compress the model size of state-of-the-art foundation Large Language Models (LLMs), as well as their instructional-finetuned variants.
- For large recommendation models such as DLRM, we propose DQRM to alleviate the cost of communications during training on distributed systems.
- Intel AI framework and oneAPI oneCCL are suitable for running the training and inference of large models.

Open-sourced Repos:
https://github.com/SqueezeAILab/SqueezeLLM
https://github.com/Zhen-Dong/BitPack
https://github.com/Zhen-Dong/HAWQ
https://github.com/Zhen-Dong/Awesome-Quantization-Papers

Thank you for listening!