Квантизация и спарсификация

Феоктистов Дмитрий, @trandelik

Материалы занятия

- Лекции Егора Швецов

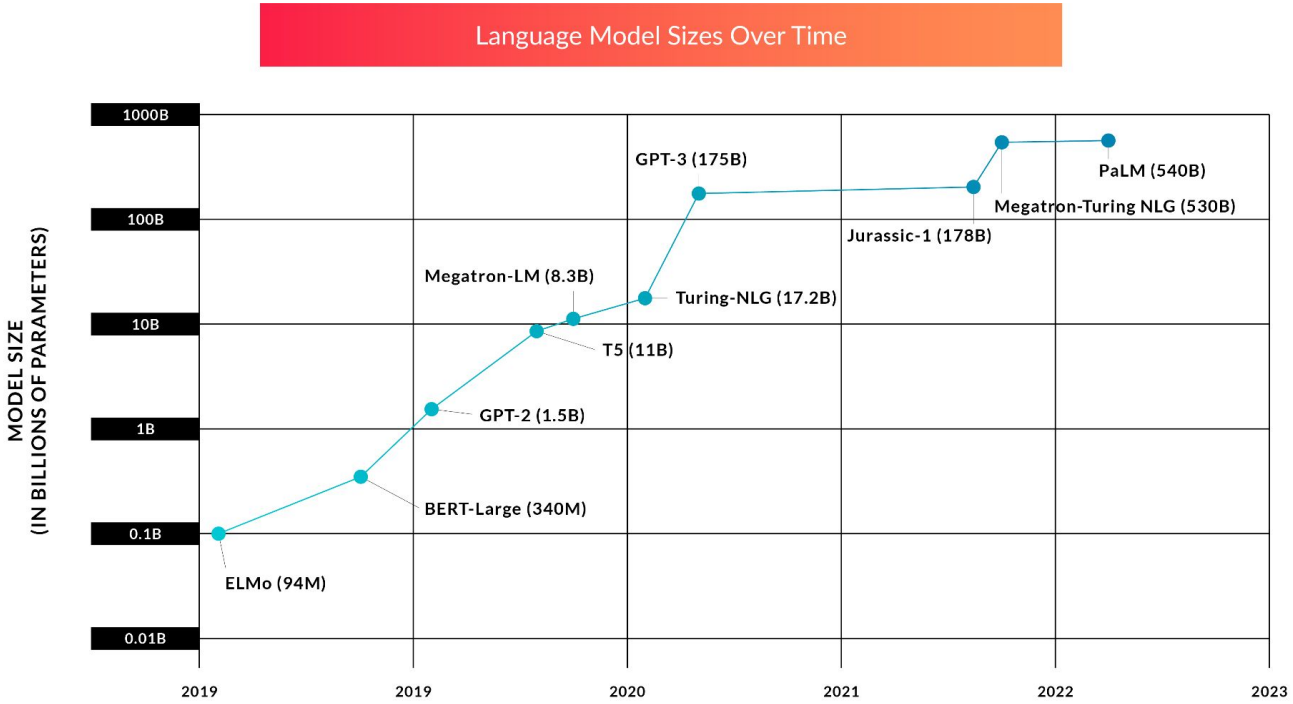
  https://github.com/On-Point-RND/Efficient-AI-Models-2024/blob/main/Week%202/Lectures/Pruning.pdf

  https://github.com/On-Point-RND/Efficient-AI-Models-2024/blob/main/Week%202/Lectures/Quatization.pdf

Квантизация

- Мотивация
- Типы данных
- Основы квантизации
- PTQ & QAT
- Квантизация LLM

Мотивация


Language Model Sizes Over Time
MODEL SIZE
(IN BILLIONS OF PARAMETERS)
1000B
100B
10B
1B
0.1B
0.01B
2019
2019
2020
2021
2022
2023
ELMo (94M)
BERT-Large (340M)
GPT-2 (1.5B)
T5 (11B)
Megatron-LM (8.3B)
Turing-NLG (17.2B)
GPT-3 (175B)
Jurassic-1 (178B)
Megatron-Turing NLG (530B)
PaLM (540B)

Типы данных

Входной тип	Тип для вычислений	Ускорение в вычислениях	Пропускная способность
FP16	FP16	8x	2x
INT8	INT8	16x	4x
INT4	INT4	32x	8x
INT1	INT1	128x	32x

Arithmetic operation	ADD	MUL
8-bit Integer	0.03 pJ	0.2 pJ
16-bit Floating point	0.4 pJ	1.1 pJ
32-bit Integer	0.1 pJ	3.1 pJ
32-bit Floating point	0.9 pJ	3.7 pJ

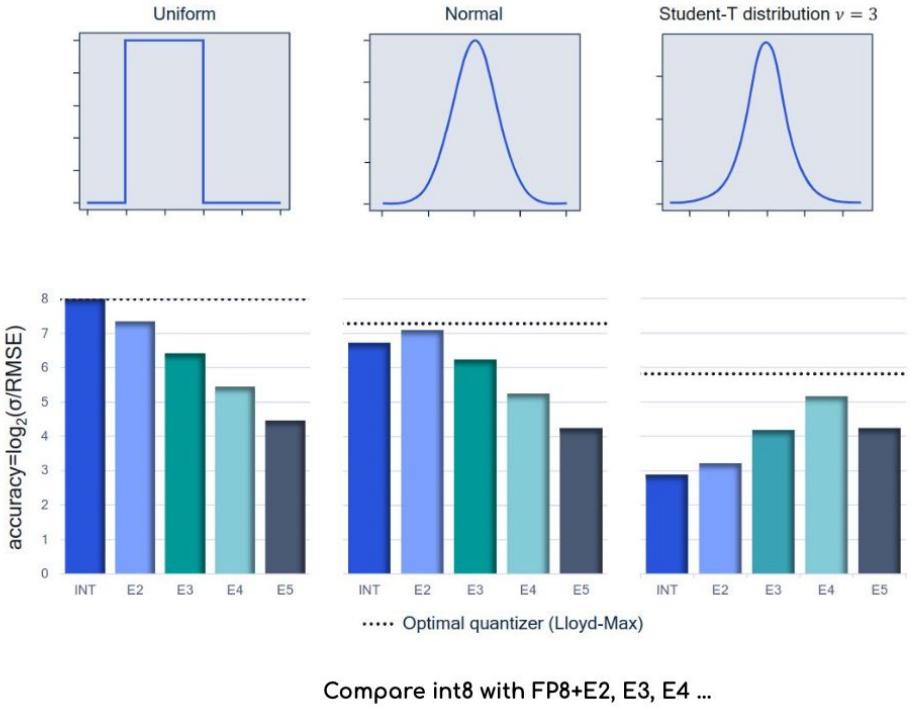
Типы данных

Входной тип	Тип для вычислений	Ускорение в вычислениях	Пропускная способность
FP16	FP16	8x	2x
INT8	INT8	16x	4x
INT4	INT4	32x	8x
INT1	INT1	128x	32x

Arithmetic operation	ADD	MUL
8-bit Integer	0.03 pJ	0.2 pJ
16-bit Floating point	0.4 pJ	1.1 pJ
32-bit Integer	0.1 pJ	3.1 pJ
32-bit Floating point	0.9 pJ	3.7 pJ

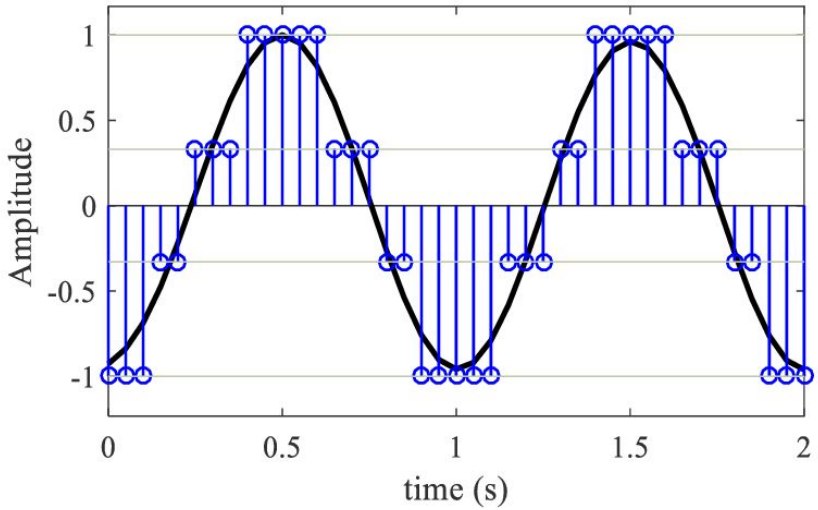
Типы данных

Инты “тупее” => дешевле в транзисторах
Флоты точнее => дороже в транзисторах
Вдруг инты не так плохи?



Compare int8 with FP8+E2, E3, E4 ...

Квантизация



Scale: $s = \frac{f_{max} - f_{min}}{fq_{max} - fq_{min}}$

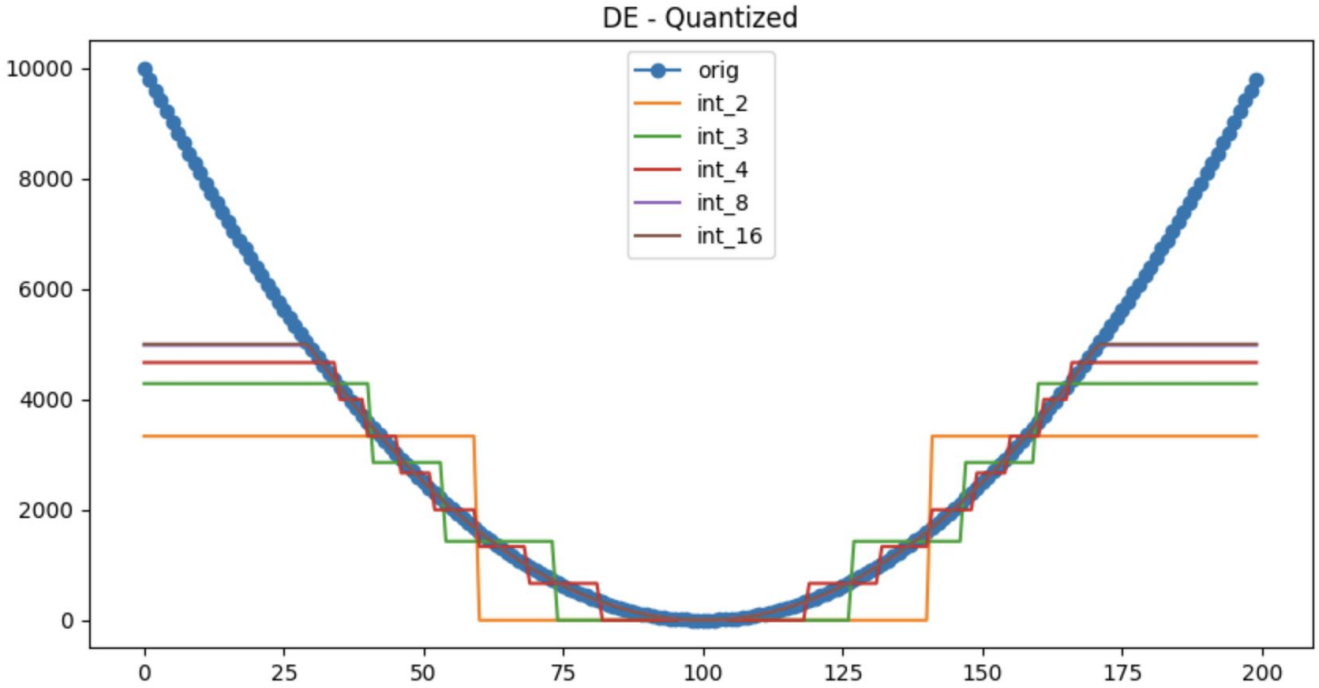
Quantize: $Q(x) = clip(round(\frac{1}{s}x), fq_{min}, fq_{max})$

De-Quantize: $x = sQ(x)$

$fq_{min} = -2^{(bit-1)}$

$fq_{max} = 2^{(bit-1)} - 1$

Квантизация

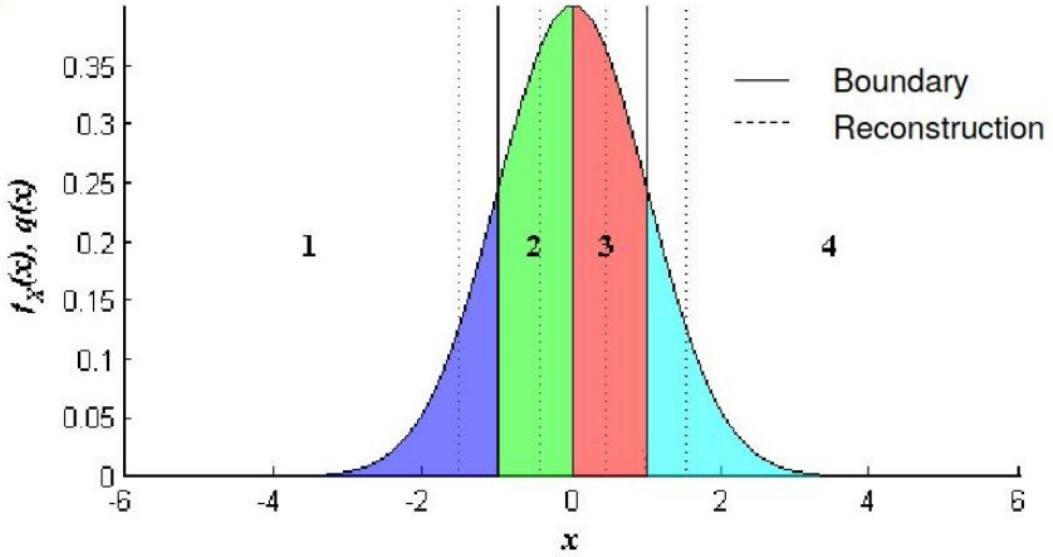


Квантизация: квантильная

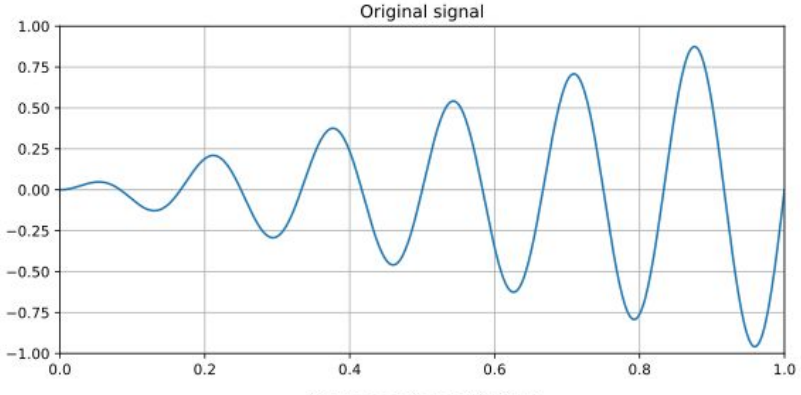
Decision thresholds -0.98, 0, 0.98

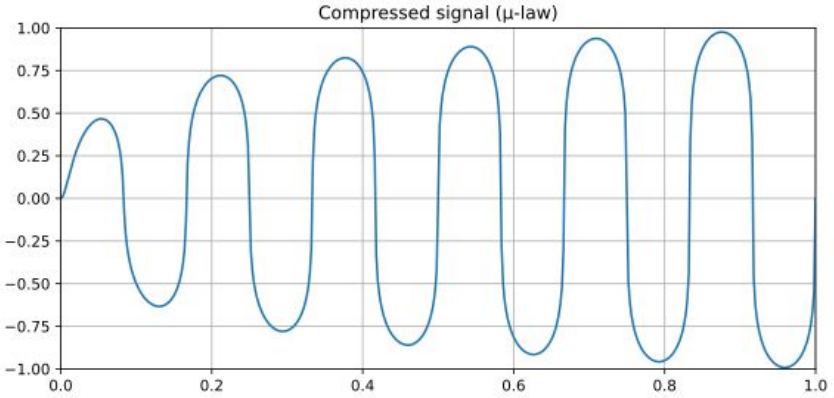
Representative levels –1.51, -0.45, 0.45, 1.51

D^*=0.12=9.30 dB



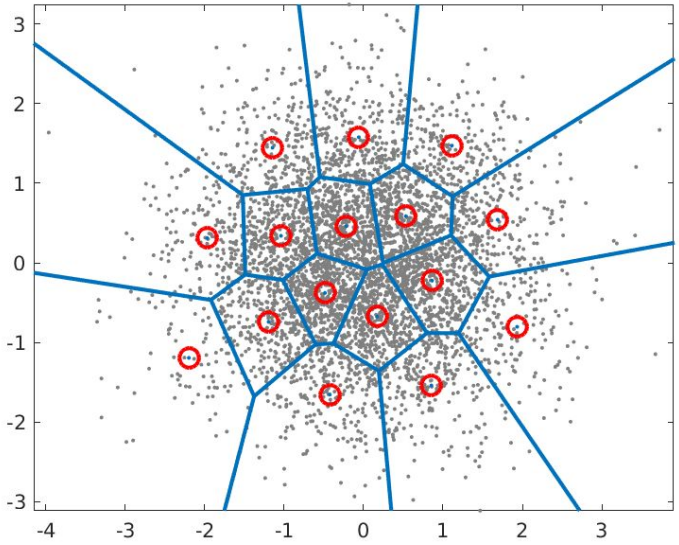
Квантизация: companding

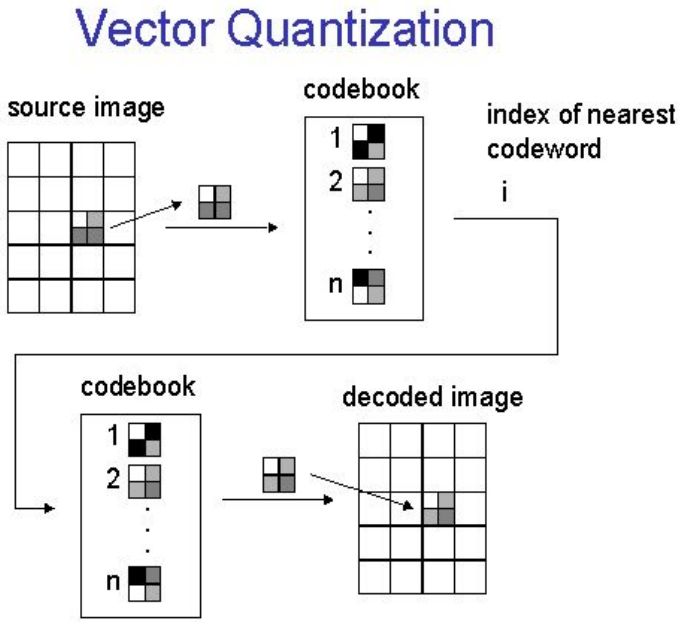




Преобразование должно быть дешевым, мы все же хотим быстрый инференс

Квантизация: векторная





\+ иерархичность

Что квантизовать?

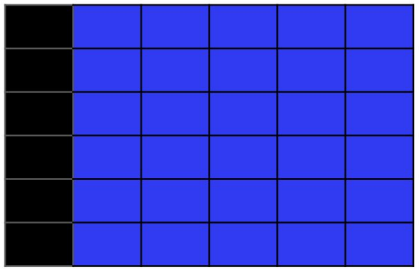
- Веса
- Веса + активации
- KV-cache

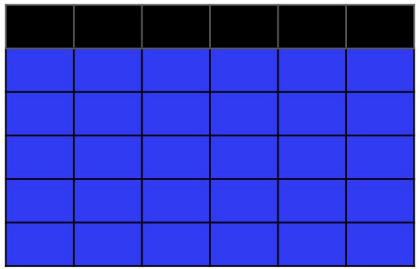
Что квантизовать?

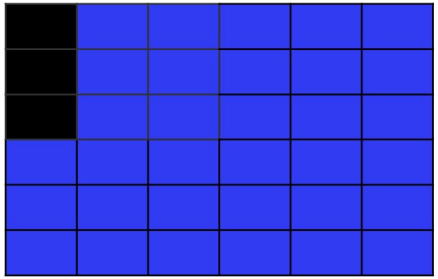
- Веса => экономим память + немного ускоряемся
- Веса + активации => экономим + ускоряемся
- KV-cache => экономим + ускоряемся

На практике нужны CUDA-кернелы

Гранулярность







QAT&PTQ

Post Training Quantization (PTQ)

1. Train a model
2. Quantize weights
3. Finetune model BN statistics or other FP blocks

PROS:
- Can be applied for already trained models
- Does not necessary require access do the data or to all the data

CONS:
- Usually has lower quality than QAT

Quantization Aware Training (QAT)

1. Quantize weights
2. Train a model with quantized weights

PROS:
- Almost lossless quality for some datasets

CONS:
- Requires approximation of non-differenitable quantization function
- Usually, requires training a model from scratch
- Need access to the data

QAT: STE

$$QW = Q(W)$$

$$y = F(x, QW)$$

$$W := W - \alpha \frac{\partial loss}{\partial QW}$$

Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation, Yoshua Bengio, 2023

QAT: DiffQ

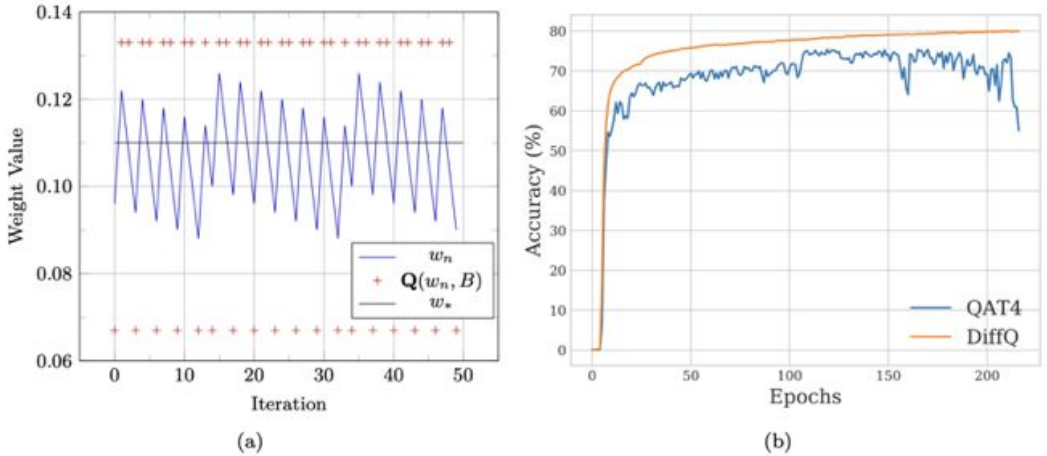


Figure 1: **(a)** Using STE and SGD to optimize the 1D least-mean-square problem given by equation 4 (with $B = 4$ and $X = 1$ a.s.). $\mathbf{Q}(w_n, B)$ oscillates between the quantized value just above (w_+) and just under (w_-) the unquantized ground truth w_*, while w_n oscillates around the boundary $(w_+ + w_-)/2$. **(b)** Model accuracy vs. epochs for ImageNet using EfficientNet-b3. Results are presented for both QAT over 4 bits and DIFFQ.

$$QW_b = Q(W, b)$$

$$QN = QW_b - W \approx \frac{\Delta_b}{2}\mathbf{U}[-1, 1]$$

$$Q\hat{W}_b = W + \frac{\Delta_b}{2}\mathbf{U}[-1, 1]$$

$$y = F(x, W + \frac{\Delta_b}{2}\mathbf{U}[-1, 1])$$

Differentiable Model Compression via Pseudo Quantization Noise, Alexandre Défossez et al, TMLR, 2022

LLM Quantization: OBQ

2 Optimal Brain Surgeon

In deriving our method we begin, as do Le Cun, Denker and Solla [1990], by considering a network trained to a local minimum in error. The functional Taylor series of the error with respect to weights (or parameters, see below) is:

$$\delta E = \left(\frac{\partial E}{\partial \mathbf{w}}\right)^T \cdot \delta \mathbf{w} + \frac{1}{2} \delta \mathbf{w}^T \cdot \mathbf{H} \cdot \delta \mathbf{w} + O(\|\delta \mathbf{w}\|^3) \tag{1}$$

where $\mathbf{H} \equiv \partial^2 E / \partial \mathbf{w}^2$ is the Hessian matrix (containing all second order derivatives) and the superscript T denotes vector transpose. For a network trained to a local minimum in error, the first (linear) term vanishes; we also ignore the third and all higher order terms. Our goal is then to set one of the weights to zero (which we call w_q) to minimize the increase in error given by Eq. 1. Eliminating w_q is expressed as:

$$\delta w_q + w_q = 0 \qquad \text{or more generally} \quad \mathbf{e}_q^T \cdot \delta \mathbf{w} + w_q = 0 \tag{2}$$

where $\mathbf{e}_q$ is the unit vector in weight space corresponding to (scalar) weight w_q. Our goal is then to solve:

$$Min_q\{Min_{\delta \mathbf{w}}\{\frac{1}{2} \delta \mathbf{w}^T \cdot \mathbf{H} \cdot \delta \mathbf{w}\} \quad \text{such that} \quad \mathbf{e}_q^T \cdot \delta \mathbf{w} + w_q = 0\} \tag{3}$$

To solve Eq. 3 we form a Lagrangian from Eqs. 1 and 2:

$$L = \frac{1}{2} \delta \mathbf{w}^T \cdot \mathbf{H} \cdot \delta \mathbf{w} + \lambda (\mathbf{e}_q^T \cdot \delta \mathbf{w} + w_q) \tag{4}$$

where λ is a Lagrange undetermined multiplier. We take functional derivatives, employ the constraints of Eq. 2, and use matrix inversion to find that the optimal weight change and resulting change in error are:

$$\delta \mathbf{w} = -\frac{w_q}{[\mathbf{H}^{-1}]_{qq}} \mathbf{H}^{-1} \cdot \mathbf{e}_q \qquad \textit{and} \qquad L_q = \frac{1}{2} \frac{w_q^2}{[\mathbf{H}^{-1}]_{qq}} \tag{5}$$

Second order derivatives for network pruning: Optimal Brain Surgeon, Babak Hassibi and David G. Stork, NIPS, 1992

LLM Quantization: OBQ

1. Train a "reasonably large" network to minimum error.
2. Compute $\mathbf{H}^{-1}$.
3. Find the q that gives the smallest saliency $L_q = w_q^2/(2[H^{-1}]_{qq})$. If this candidate error increase is much smaller than E, then the q^{th} weight should be deleted, and we proceed to step 4; otherwise go to step 5. (Other stopping criteria can be used too.)
4. Use the q from step 3 to update *all* weights (Eq. 5). Go to step 2.
5. No more weights can be deleted without large increase in E. (At this point it may be desirable to retrain the network.)

Second order derivatives for network pruning: Optimal Brain Surgeon, Babak Hassibi and David G. Stork, NIPS, 1992

LLM Quantization: OBQ

The Quantization Order and Update Derivations. Under the standard assumption that the gradient at the current point $\mathbf{w}$ is negligible, the OBS formulas for the optimal weight to be pruned w_p and the corresponding update $\boldsymbol{\delta_p}$ can be derived by writing the locally quadratic problem under the constraint that element p of $\boldsymbol{\delta_p}$ is equal to $-w_p$, which means that w_p is zero after applying the update to $\mathbf{w}$. This problem has the following Lagrangian:

$$L(\boldsymbol{\delta_p}, \lambda) = \boldsymbol{\delta}_\mathbf{p}^\top \mathbf{H} \boldsymbol{\delta_p} + \lambda(\mathbf{e}_\mathbf{p}^\top \boldsymbol{\delta_p} - (-w_p)), \tag{6}$$

where $\mathbf{H}$ denotes the Hessian at $\mathbf{w}$ and $\mathbf{e_p}$ is the pth canonical basis vector. The optimal solution is then derived by first finding the optimal solution to $\boldsymbol{\delta_p}$ via setting the derivative $\partial L / \partial \boldsymbol{\delta_p}$ to zero and then substituting this solution back into L and solving for λ; please see e.g. [13, 39] for examples.

Assume a setting in which we are looking to quantize the weights in a layer on a fixed grid of width Δ while minimizing the loss. To map OBS to a *quantized* projection, we can set the target of the Lagrangian constraint in (6) to $(\text{quant}(w_p) - w_p)$, where $\text{quant}(w_p)$ is the weight rounding given by quantization; then $w_p = \text{quant}(w_p)$ after the update.

Assuming we wish to quantize weights iteratively, one-at-a-time, we can derive formulas for the "optimal" weight to quantize at a step, in terms of minimizing the loss increase, and for the corresponding optimal update to the unquantized weights, in similar fashion as discussed above:

$$w_p = \text{argmin}_{w_p} \frac{(\text{quant}(w_p) - w_p)^2}{[\mathbf{H}^{-1}]_{pp}}, \quad \boldsymbol{\delta_p} = -\frac{w_p - \text{quant}(w_p)}{[\mathbf{H}^{-1}]_{pp}} \cdot \mathbf{H}^{-1}_{:,p}. \tag{7}$$

In fact, since $-w_p$ is a constant during all derivations, we can just substitute it with $(\text{quant}(w_p) - w_p)$ in the final result. We note that the resulting formulas are a generalization of standard OBS for pruning, if $\text{quant}(\cdot)$ always "quantizes" a weight to 0, then we recover the original form.

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning, Elias Frantar et al, NeurIPS, 2022

LLM Quantization: OBQ

Lemma 1 (Row & Column Removal). *Given an invertible $d_{col} \times d_{col}$ matrix $\mathbf{H}$ and its inverse $\mathbf{H}^{-1}$, we want to efficiently compute the inverse of $\mathbf{H}$ with row and column p removed, which we denote by $\mathbf{H}_{-p}$. This can be accomplished through the following formula:*

$$\mathbf{H}_{-p}^{-1} = \left(\mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{pp}} \mathbf{H}^{-1}_{:,p} \mathbf{H}^{-1}_{p,:}\right)_{-p}, \tag{4}$$

which corresponds to performing Gaussian elimination of row and column p in $\mathbf{H}^{-1}$ followed by dropping them completely. This has $\Theta(d_{col}^2)$ time complexity.

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning, Elias Frantar et al, NeurIPS, 2022

LLM Quantization: OBQ

We follow these conventions as well, and work with the specific layer-wise compression problem stated formally below, where the weights $\mathbf{W}_\ell$ are a $d_{\text{row}} \times d_{\text{col}}$ matrix (for conv-layers d_{col} corresponds to the total number of weights in a single filter), and the input $\mathbf{X}_\ell$ has dimensions $d_{\text{col}} \times N$.

$$\text{argmin}_{\widehat{\mathbf{W}}_\ell} \quad ||\mathbf{W}_\ell \mathbf{X}_\ell - \widehat{\mathbf{W}}_\ell \mathbf{X}_\ell||_2^2 \quad \text{s.t.} \quad \mathcal{C}(\widehat{\mathbf{W}}_\ell) > C. \tag{2}$$

+ улучшения и трюки
  много трюков

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning, Elias Frantar et al, NeurIPS, 2022

LLM Quantization: GPT-Q

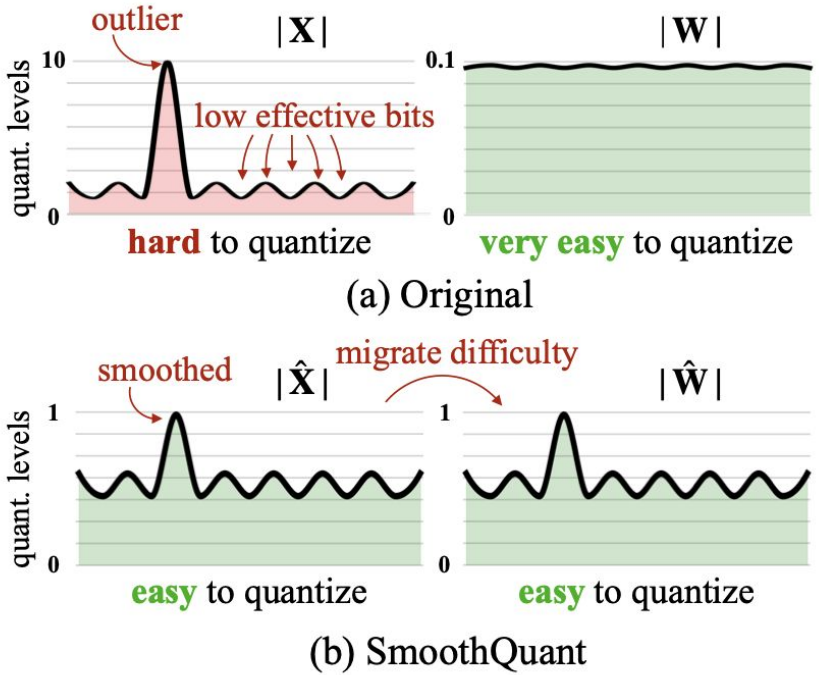
Algorithm 1 Quantize $\mathbf{W}$ given inverse Hessian $\mathbf{H}^{-1} = (2\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}$ and blocksize B.

$\mathbf{Q} \leftarrow \mathbf{0}_{d_{\text{row}} \times d_{\text{col}}}$ *// quantized output*
$\mathbf{E} \leftarrow \mathbf{0}_{d_{\text{row}} \times B}$ *// block quantization errors*
$\mathbf{H}^{-1} \leftarrow \text{Cholesky}(\mathbf{H}^{-1})^\top$ *// Hessian inverse information*
for $i = 0, B, 2B, \ldots$ **do**
  for $j = i, \ldots, i + B - 1$ **do**
    $\mathbf{Q}_{:,j} \leftarrow \text{quant}(\mathbf{W}_{:,j})$ *// quantize column*
    $\mathbf{E}_{:,j-i} \leftarrow (\mathbf{W}_{:,j} - \mathbf{Q}_{:,j}) \,/\, [\mathbf{H}^{-1}]_{jj}$ *// quantization error*
    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{W}_{:,j:(i+B)} - \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$ *// update weights in block*
  end for
  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{W}_{:,(i+B):} - \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$ *// update all remaining weights*
end for

1) Arbitrary Order Insight => можем квантовать все строки одновременно => сэкономим на гессианах (посчитаем их заранее параллельно)
2) Memory Bound => Batching по столбцам => сэкономим на компьюте
3) Разложение Холецкого вместо полного обращения гессиана => быстрее, стабильнее, чем исключение строк + много нулей
4) **Кернелы**

GPTQ: Accurate Post-Training Quantization For Generative Pre-Trained Transformers, Elias Frantar et al, ICLR 2023

LLM Quantization: Outliers



SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models, Guangxuan Xiao et al, ICML, 2023

LLM Quantization: Outliers

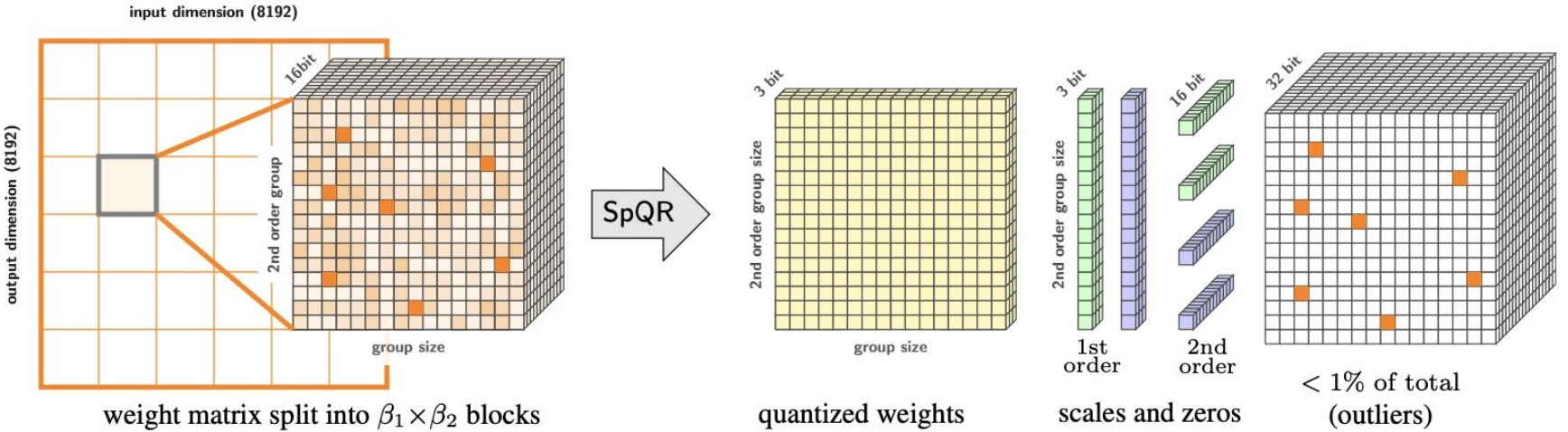


Figure 3: A high-level overview of the SpQR representation for a single weight tensor. The right side of the image depicts all stored data types and their dimensions.

SPQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression, Tim Dettmers et al, ICLR 2024

LLM Quantization: Scaling Laws

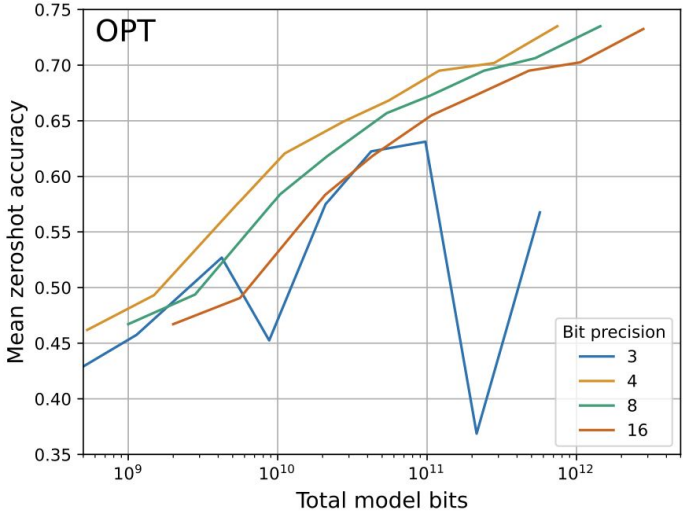


Figure 1. Bit-level scaling laws for mean zero-shot performance across four datasets for 125M to 176B parameter OPT models. Zero-shot performance increases steadily for fixed model bits as we reduce the quantization precision from 16 to 4 bits. At 3-bits, this relationship reverses, making 4-bit precision optimal.

We run more than 35,000 experiments with 16-bit inputs and k-bit parameters to examine which zero-shot quantization methods improve scaling for 3 to 8-bit precision at scales of 19M to 176B parameters across the LLM families BLOOM, OPT, NeoX/Pythia, and GPT-2. We find that it is challenging to improve the bit-level scaling trade-off, with the only improvements being the use of a small block size – splitting the parameters into small independently quantized blocks – and the quantization data type being used

The case for 4-bit precision: k-bit Inference Tim Dettmers, Luke Zettlemoyer, ICML, 2023

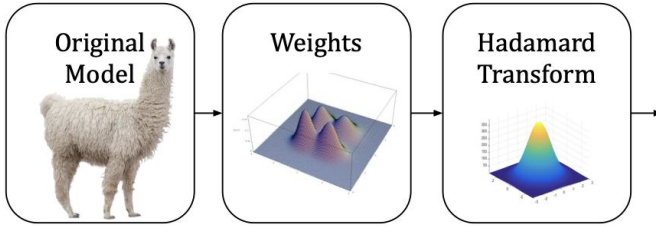
AQLM vs QUIP#

Algorithm 1 AQLM: Additive Quantization for LLMs

Require: model, data

$\mathbf{X}_{block} :=$ `model.input_embeddings(data)`
for $i = 1, \ldots,$ `model.num_layers` **do**
  `block := model.get_block(i)`
  $\mathbf{Y}_{block} :=$ `block`$(\mathbf{X}_{block})$
  for `layer` $\in$ `linear_layers(block)` **do**
    $\mathbf{W} :=$ `layer.weight`
    $\mathbf{X} :=$ `layer_inputs(layer,`$\mathbf{X}_{block}$`)`
    $C, b, s :=$ `initialize`$(\mathbf{W})$ `// k-means`
    while loss improves by at least τ **do**
      $C, s :=$ `train_Cs_adam`$(\mathbf{X}\mathbf{X}^T, \mathbf{W}, C, b, s)$
      $b :=$ `beam_search`$(\mathbf{X}\mathbf{X}^T, \mathbf{W}, C, b, s)$
    end while
    `/* save for fine-tuning */`
    `layer.weight :=` `AQLMFormat`(C, b, s)
  end for
  $\theta :=$ `trainable_parameters(block)`
  while loss improves by at least τ **do**
    $L := ||$`block`$(\mathbf{X}_{block}) - \mathbf{Y}_{block}||_2^2$
    $\theta :=$ `adam`$(\theta, \frac{\partial L}{\partial \theta})$
  end while
  $\mathbf{Y}_{block} :=$ `block`$(\mathbf{X}_{block})$
end for

январь 2024



Definition 1. *We say a symmetric Hessian matrix $H \in \mathbb{R}^{n\times n}$ is μ-incoherent if it has an eigendecomposition $H = Q\Lambda Q^T$ such that for all i and j, $|Q_{ij}| = \left|e_i^T Q e_j\right| \leq \mu/\sqrt{n}$. By extension, we say a weight matrix $W \in \mathbb{R}^{m\times n}$ is μ-incoherent if all i and j, $|W_{ij}| = \left|e_i^T W e_j\right| \leq \mu \|W\|_F / \sqrt{mn}$.*

Algorithm 3 QuIP: Quantization with Incoherence Processing

Require: $b \in \mathbb{N}$, $H \in \mathbb{R}^{n\times n}$ SPD, $W \in \mathbb{R}^{m\times n}$, $\mathcal{Q} \in \{\mathsf{Near}, \mathsf{Stoch}\}$, $\rho \in \mathbb{R}_+$, $\alpha \in [0, 1]$

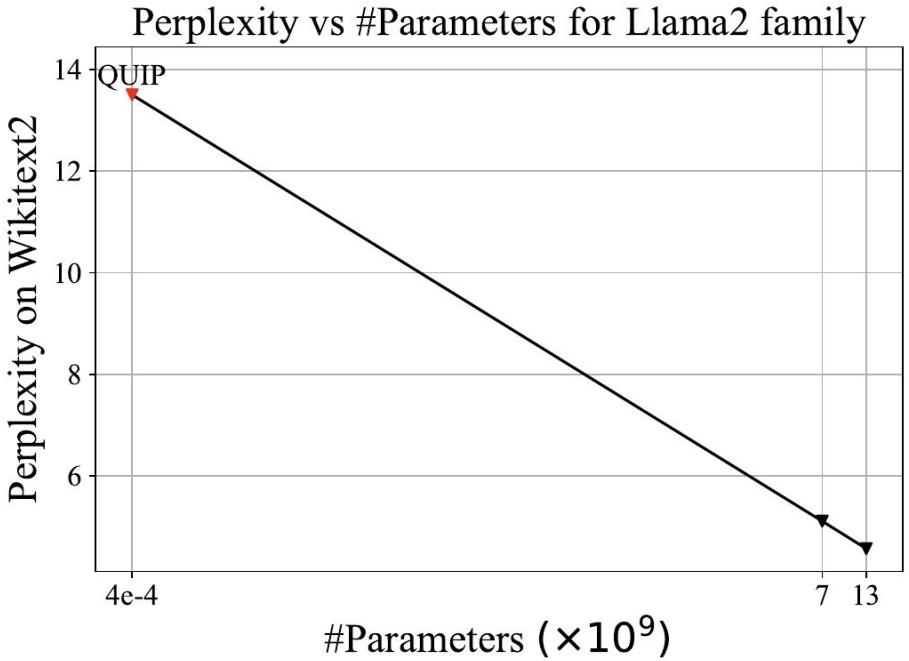
$\hat{W}, H, s, \tilde{D} \leftarrow$ Alg 1(b, H, W, ρ, α) ▷ QuIP Incoherence Pre-Procesing
$H = (\dot{U} + I)D(\dot{U} + I)^{-1}$ ▷ LDL decomposition
for $k \in \{1, \ldots, n\}$ **do** $\hat{W}_k \leftarrow \mathrm{clamp}(\mathcal{Q}(W_k + (W - \hat{W})\dot{U}_k), 0, 2^b - 1)$ ▷ LDLQ
return $\hat{W} \leftarrow$ Alg 2$(b, H, \hat{W}, s, \tilde{D})$ ▷ QuIP Incoherence Post-Processing

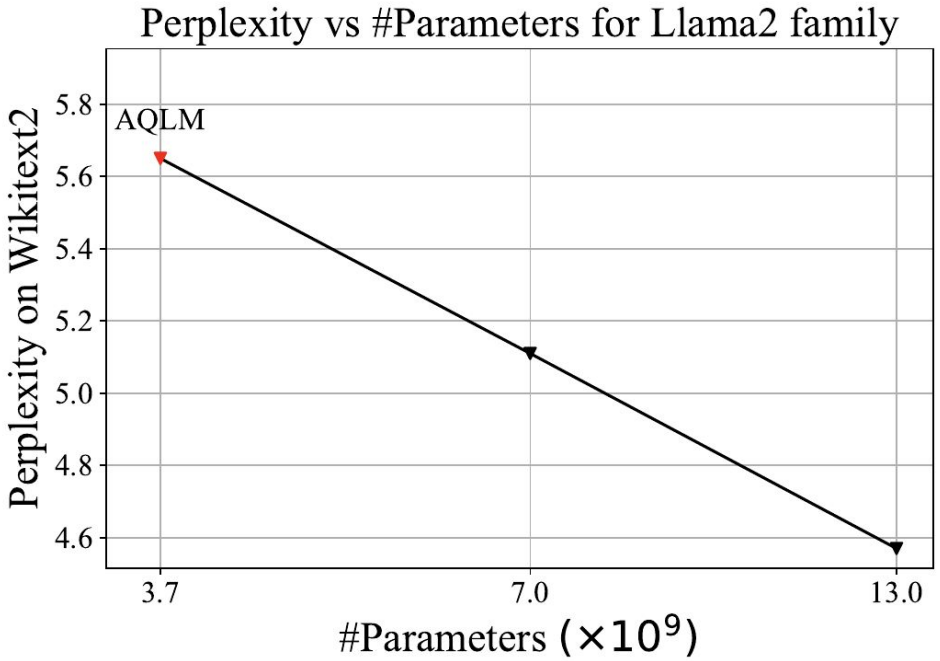
середина 2023

Extreme Compression of Large Language Models via Additive Quantization, Egiazarian et al, ICML, 2024

QUIP: 2-Bit Quantization of Large Language Models With Guarantees, Jerry Chee et al, NeurIPS, 2023

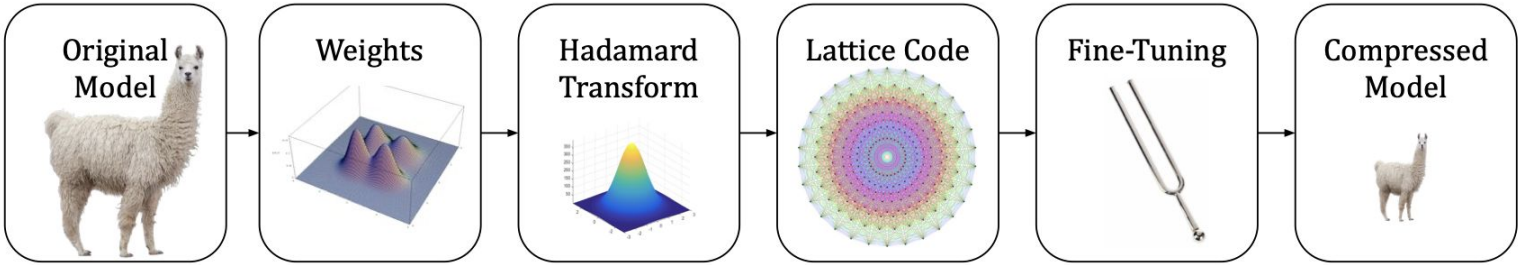
AQLM vs QUIP#



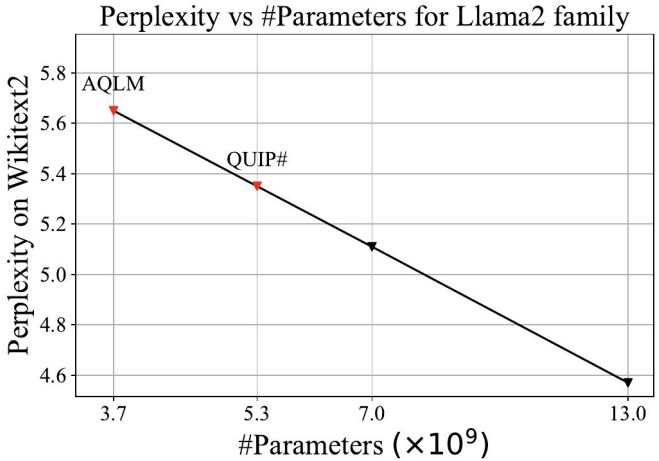


Extreme Compression of Large Language Models via Additive Quantization, Egiazarian et al, ICML, 2024
QUIP: 2-Bit Quantization of Large Language Models With Guarantees, Jerry Chee et al, NeurIPS, 2023

AQLM vs QUIP#



февраль 2024



QuIP#: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks, Albert Tseng et al, ICML, 2024

AQLM vs QUIP#

Algorithm 1 PV algorithm

Initialization: starting point $x^0 \in \mathbb{R}^d_{\leq c}$
for $k = 0, 1, \ldots$ **do**
  $y^k = M_P(x^k) := \arg\min_{y \in \mathbb{R}^d} \left\{ \phi(y) : P(y) \supseteq P(x^k) \right\}$ (P step: continuous)
  $x^{k+1} = M_V(y^k) := \arg\min_{x \in \mathbb{R}^d} \left\{ \phi(x) : V(x) \subseteq V(y^k) \right\}$ (V step: discrete)
end for

Algorithm 2 PV-Tuning: Optimization

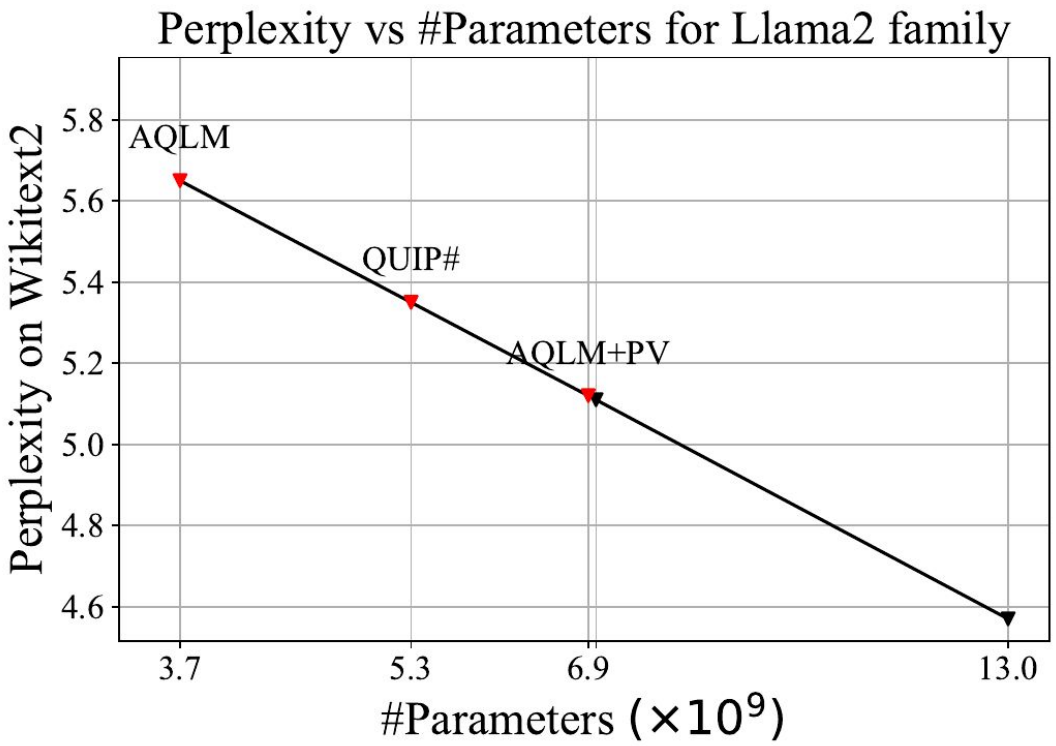
Require: initial parameters $x^0 \in \mathbb{R}^d_c$,
objective function $\phi : \mathbb{R}^d \to \mathbb{R}$,
subspace size $\tau \in [d]$

for $k = 0, \ldots, K-1$ **do**
  ▷ **P step:** update $V(x)$ by backprop
  $y^k = \arg\min_{y \in \mathbb{R}^d_{\leq c}} \{ \phi(y) \ : \ P(y) \supseteq P(x^k) \}$
  ▷ **V step:** choose a subspace $\mathcal{S}^k$ & update $P(x)$
  $\mathcal{S}^k = \arg\operatorname{top}_{1 \leq i \leq d} \tau \ \ |\nabla_i \phi(y^k)|$ ▷ find τ largest
  $\widehat{\phi}_{y,\mathcal{S}^k}(x) := \left\| x - \left(y - \frac{1}{L_{\mathcal{S}^k}} Z^k \left(\nabla \phi(y) \right) \right) \right\|^2$
  $x^{k+1} \in \arg\min_x \left\{ \widehat{\phi}_{y^k,\mathcal{S}^k}(x) : V(x) \subseteq V(y^k) \right\}$
end for

май 2024

PV-tuning: Beyond Straight-Through Estimation for Extreme LLM Compression, Vladimir Malinovskii et al, NeurIPS, 2024

AQLM vs QUIP#


Perplexity vs #Parameters for Llama2 family
Perplexity on Wikitext2
5.8
5.6
5.4
5.2
5.0
4.8
4.6
AQLM
QUIP#
AQLM+PV
3.7
5.3
6.9
13.0
#Parameters (×10⁹)

AQLM vs QUIP#

AQLM

Dettmers & Zettlemoyer (2022) examined the accuracy-compression trade-offs of these early methods, suggesting that 4-bit quantization may be optimal for RTN quantization, and observing that data-aware methods like GPTQ allow for higher compression, i.e. strictly below 4 bits/weight, maintaining Pareto optimality. Our work brings this Pareto frontier below 3 bits/weight, for the first time. Parallel

AQLM+PV

Pareto-optimality. A key practical question concerns obtaining optimal quality for the target model size, where a smaller model compressed to 3-4 bits often dominates a larger model compressed to 1-bit. The best known Pareto-optimal bit-width for Llama 2 is 2.5 [19]: compressing a larger model to less than 2.5 bits per weight is inferior to a smaller model quantized to the same total number of bytes. From this perspective, **PV-tuning pushes the Pareto-optimal frontier for LLAMA 2 to 2.0 bits**. This is easiest to see in Table 8: a 2-bit 13B model outperforms any 7B quantization and is comparable with the 16-bit 7B model. The same holds for the 2-bit 70B model.

KV-cache quantization

The core idea of AQUA-KV is to leverage inter-dependencies between consecutive KV-caches to improve compression. **For this, we train compact predictors that "guess" the value of a Key & Value pair using other cache entries, then quantize the residual information that could not be predicted.** This way, we only store the information that cannot be recovered from other sources.

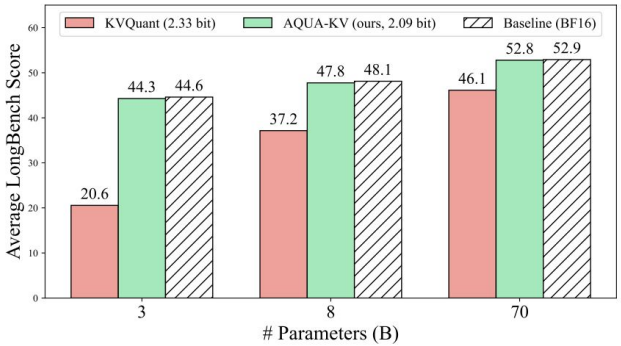


Figure 1. Comparison of AQUA-KV to alternative Key-Value Cache compression methods for Llama 3.x models in terms of average LongBench score on 14 english tasks (see Section 4).

Algorithm 1 AQUA-KV Calibration

Require: model, data, quantization method Q

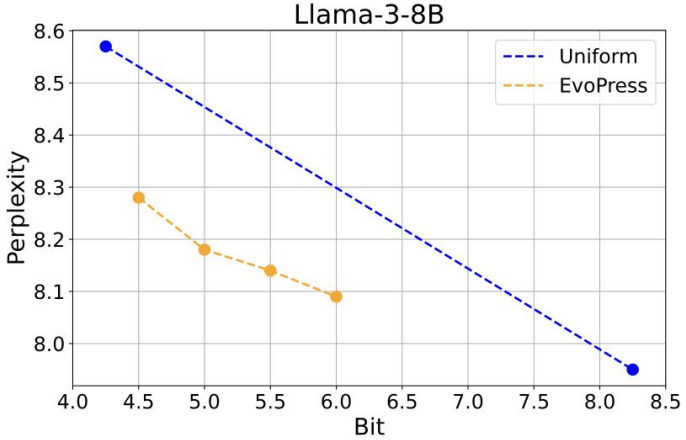
$\mathbf{X} \leftarrow$ `model.input_embeddings(data)`
$\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \emptyset$
`predictors` $\leftarrow \{\}$
for $i = 1, \ldots,$ `model.num_layers` **do**
  `block` $\leftarrow$ `model.transformer_layers`$[i]$
  $(\mathbf{K}, \mathbf{V}) \leftarrow$ `block.get_attention_kv`$(\mathbf{X})$
  $\mathbf{X} \leftarrow$ `block`$(\mathbf{X})$
  if $\mathbf{K}_{\text{old}} = \emptyset$ **and** $\mathbf{V}_{\text{old}} = \emptyset$ **then**
    $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}, \mathbf{V}$
    continue
  end if
  # Predict keys from past keys
  $f_{\text{key}} \leftarrow \arg\min_f \|f(\mathbf{K}_{\text{old}}) - \mathbf{K}\|_2^2$
  $\mathbf{K}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{K} - f_{\text{key}}(\mathbf{K}_{\text{old}})))$
  $\mathbf{K}_{\text{rec}} \leftarrow f_{\text{key}}(\mathbf{K}_{\text{old}}) + \mathbf{K}_{\text{residue}}$
  # Predict values from past values and current keys
  $f_{\text{value}} \leftarrow \arg\min_f \|f([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) - \mathbf{V}\|_2^2$
  $\mathbf{V}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{V} - f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}])))$
  $\mathbf{V}_{\text{rec}} \leftarrow f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) + \mathbf{V}_{\text{residue}}$
  `predictors[i]` $= (f_{\text{key}}, f_{\text{value}})$
  $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}_{\text{rec}}, \mathbf{V}_{\text{rec}}$
end for
return `predictors`

Cache Me If You Must: Adaptive Key-Value Quantization for Large Language Models, Alina Shutova et al, ICML, 2025

Квантизация в смешанную битность

$$\hat{M}^* = \arg\max_{\hat{M}} f(\hat{M}) \quad \text{subject to} \quad g(\hat{M}) \leq C,$$

where $f(\hat{M})$ quantifies the performance of the compressed model $\hat{M}$ and $g(\hat{M})$ represents the compression constraint. For simplicity, we will define g as the model's total size (in terms of parameters); however, the method can be adapted to other constraints, such as inference speed.



Algorithm 1: EvoPress: A $(1+\lambda)$-Evolutionary Algorithm with Level-Switch Mutation and Multi-Step Selection for Maximizing $f : [m]^n \to \mathbb{R}$.

Initialization: `candidates` $\leftarrow []$;
for $i \leftarrow 1$ **to** *initialCandidates* **do**
  `candidate` $\leftarrow$ `sampleUniformly()`;
  `candidates.append(candidate)`;
$x^{(1)} \leftarrow$ `selectTopKFittest(candidates, initialTokens,` $K = 1$`)`;
Optimization: **for** $t \leftarrow 1$ **to** ∞ **do**
  `offspring` $\leftarrow []$;
  Mutation: **for** $i \leftarrow 1$ **to** λ **do**
    $y_i \leftarrow x^{(t)}$;
    $y_i \leftarrow$ `LevelSwitchMutation`(y_i);
    `offspring.append`(y_i);
  Selection: **for** *step* $\leftarrow 1$ **to** *selectSteps* **do**
    Elitism: **if** *step* $=$ *selectSteps* **then**
      `offspring.append`$(x^{(t)})$;
    `offs.` $\leftarrow$ `selectTopKFittest(offs., tokens[step],` $K =$ `survivors[step])`;
  $x^{(t+1)} \leftarrow$ `offspring[0]`;

EvoPress: Towards Optimal Dynamic Model Compression via Evolutionary Search, Oliver Sieberling et al, ICML, 2025

Стабилизация обучения в низкой битности

1) Делаем QAT
2) Возьмем SOTA приемы из обычной квантизации (Hadamar Transform, подбор скейлинга, etc)
3) Нужно улучшить оценку градиентов
4) Давайте оставим только градиенты от параметров, на которых ошибка квантизации маленькая!
5) В какой-то степени эквивалентно обучению с сжатыми градиентами, такое оптимизация ест

Then, the squared gradient difference in (2) decomposes as:

$$\underbrace{\sum_{k \in S_{\text{small}}} (\nabla_{\mathbf{w}} L_k \; - \; \nabla_{\widehat{\mathbf{w}}} L_k)^2}_{(\star)} + \underbrace{\sum_{k \in S_{\text{large}}} (\nabla_{\mathbf{w}} L_k \; - \; \nabla_{\widehat{\mathbf{w}}} L_k)^2}_{(\star\star)}.$$

Assuming that the loss L is γ-smooth, the $(\star)$ "small error" term would be upper bounded by $\gamma^2 T^2 |S_{\text{small}}|$. Intu-

To bound the second term $(\star\star)$, we choose to *not trust* the gradient estimations for weights with large errors $\{\nabla_{\widehat{\mathbf{w}}} L_k : k \in S_{\text{large}}\}$. Choosing $T = \frac{\alpha^*}{2^b - 1}$ and masking gradients for elements in S_{large} we obtain the gradient operator:

$$\frac{\partial}{\partial \mathbf{x}} \approx \mathbf{I}_{|\hat{\mathbf{x}} - \mathbf{x}| \leq T} \odot \frac{\partial}{\partial \hat{\mathbf{x}}} := M_{\alpha^*}(\mathbf{x}; \hat{\mathbf{x}}) \odot \frac{\partial}{\partial \hat{\mathbf{x}}},$$

where $\mathbf{I}_{|\hat{\mathbf{x}} - \mathbf{x}| \leq T}$ is the standard indicator operator. We will refer to M_{α^*} as the "trust mask"; this gradient estimation operator will be called the **trust estimator**.

QuEST Stable Training of LLMs with 1-Bit Weights and Activations, Andrei Panferov et al, ICML, 2025

Стабилизация обучения в низкой битности

Algorithm 1 `QuEST Training Forward`

Input: Input activations $\mathbf{x}$, row-major weight $\mathbf{w}$

$\mathbf{x}_h = \mathrm{HT}(\mathbf{x})$

$\hat{\mathbf{x}}_h = \mathrm{proj}_{\alpha^*}\, \mathbf{x}_h$

$\mathbf{w}_h = \mathrm{HT}(\mathbf{w})$

$\hat{\mathbf{w}}_h = \mathrm{proj}_{\alpha^*}\, \mathbf{w}_h$

$\mathbf{y} = \hat{\mathbf{x}}_h \hat{\mathbf{w}}_h^T$

Return: $\mathbf{y}, \hat{\mathbf{x}}_h, \hat{\mathbf{w}}_h, M_{\alpha^*}(\mathbf{x}_h; \hat{\mathbf{x}}_h), M_{\alpha^*}(\mathbf{w}_h; \hat{\mathbf{w}}_h)$

Algorithm 2 `QuEST Training Backward`

Input: $\frac{\partial L}{\partial \mathbf{y}}, \hat{\mathbf{x}}_h, \hat{\mathbf{w}}_h, M_{\alpha^*}(\mathbf{x}_h; \hat{\mathbf{x}}_h), M_{\alpha^*}(\mathbf{w}_h; \hat{\mathbf{w}}_h)$

$\frac{\partial L}{\partial \hat{\mathbf{x}}_h} = \frac{\partial L}{\partial \mathbf{y}} \hat{\mathbf{w}}_h$

$\frac{\partial L}{\partial \mathbf{x}} = \mathrm{IHT}\left(M_{\alpha^*}(\mathbf{x}_h; \hat{\mathbf{x}}_h) \odot \frac{\partial L}{\partial \hat{\mathbf{x}}_h} \right)$

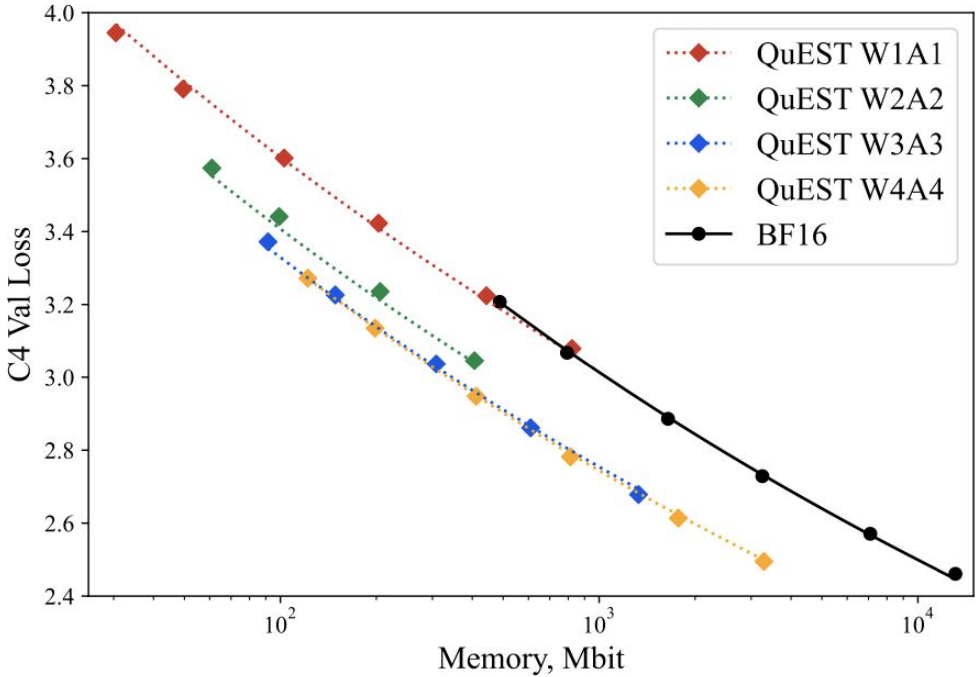
$\frac{\partial L}{\partial \hat{\mathbf{w}}_h} = \hat{\mathbf{x}}_h^T \frac{\partial L}{\partial \mathbf{y}}$

$\frac{\partial L}{\partial \mathbf{w}} = \mathrm{IHT}\left(M_{\alpha^*}(\mathbf{w}_h; \hat{\mathbf{w}}_h) \odot \frac{\partial L}{\partial \hat{\mathbf{w}}_h} \right)$

Return: $\frac{\partial L}{\partial \mathbf{x}}, \frac{\partial L}{\partial \mathbf{w}}$

QuEST Stable Training of LLMs with 1-Bit Weights and Activations, Andrei Panferov et al, ICML, 2025

Стабилизация обучения в низкой битности

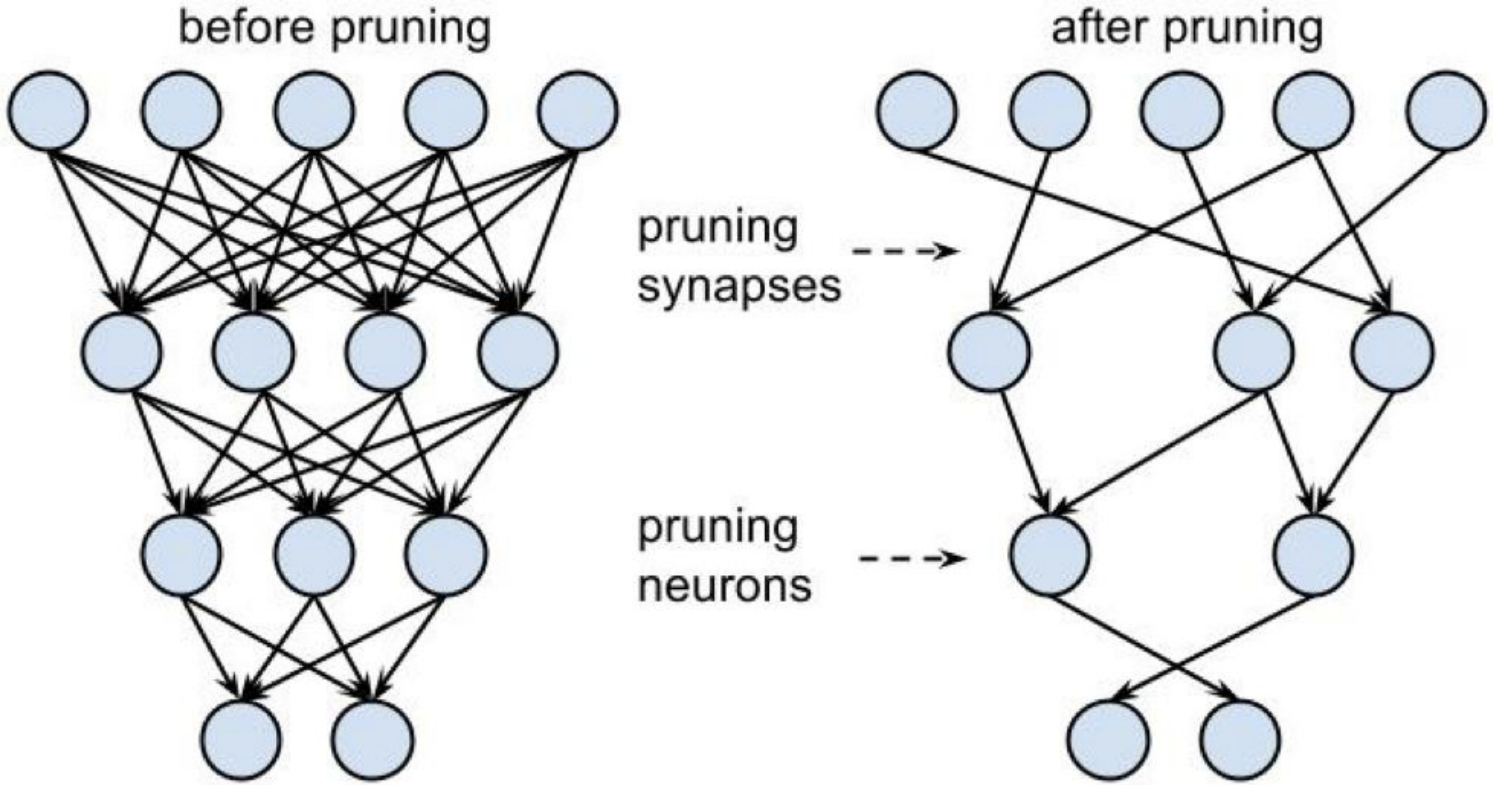


QuEST Stable Training of LLMs with 1-Bit Weights and Activations, Andrei Panferov et al, ICML, 2025

Спарсификация

- TPF, PAI, PTP, TTP
- Structured & Unstructured
- Основы квантизации
- LLM Pruning
- Pruning vs Quantization

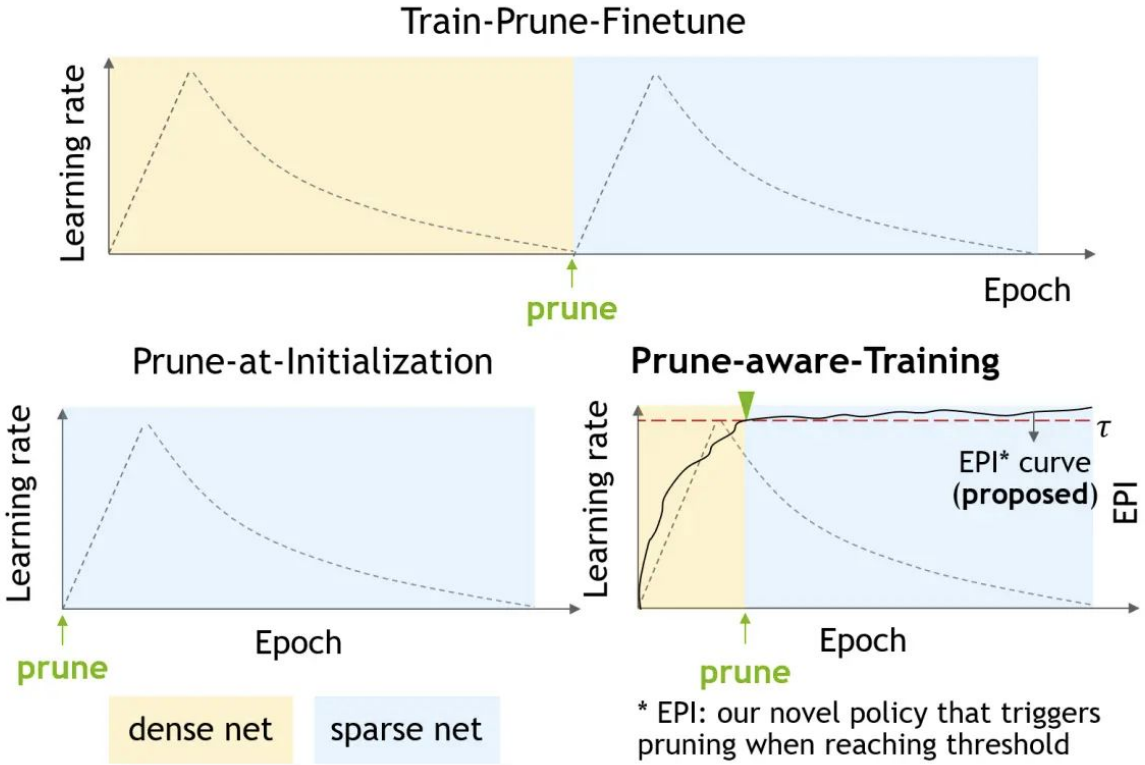
Общая идея


before pruning
after pruning
pruning
synapses
pruning
neurons

Общая идея

It is well documented that neural networks have an excess of parameters needed to generalize well and make accurate predictions. *“The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks” (Frankle and Carbin, 2019)* demonstrates that **neural networks tend to have a specific subset of parameters that are essential for prediction**. Model pruning is a very intuitive approach to model compression, with the hypothesis that effective model compression should **remove weights that aren’t being used**, similarly to how the brain reduces usage of connections between neurons to emphasize important pathways.

TPF, PAI, PTP, PAT



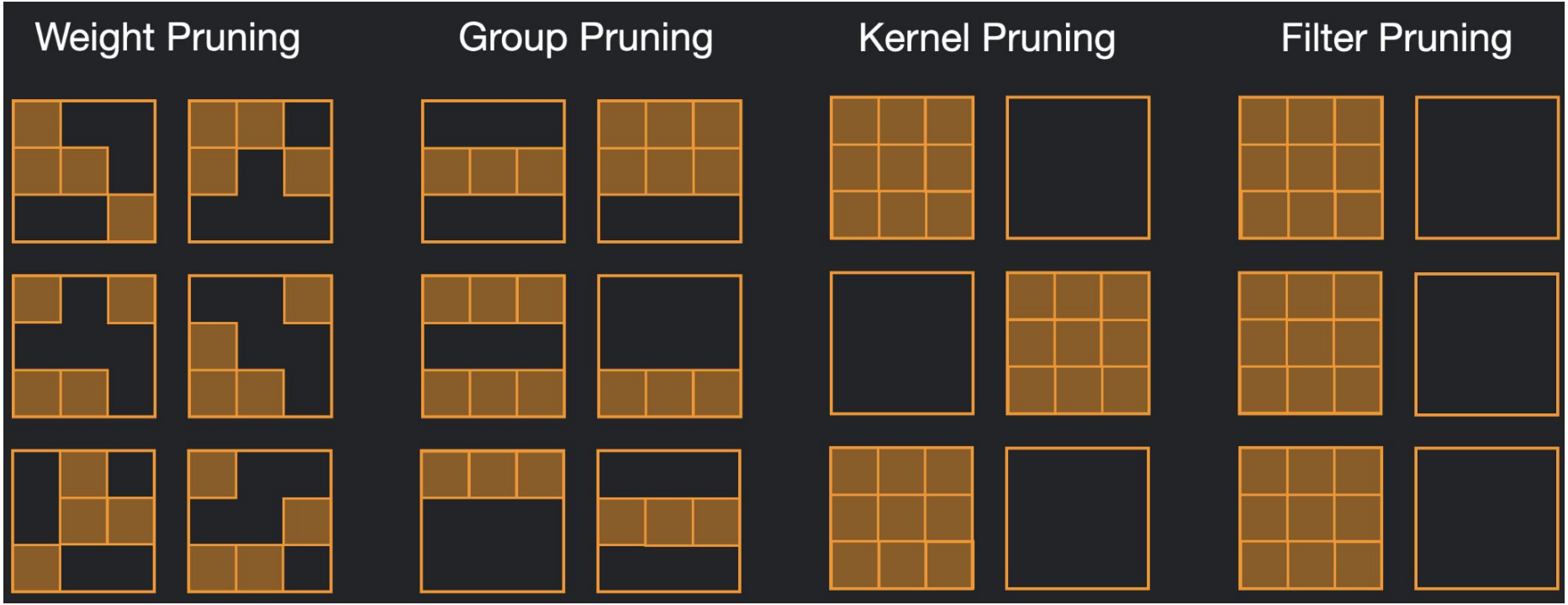
TPF, PAI, PTP, PAT

Datature | Blog

PRUNING APPROACH	TRAIN-TIME PRUNING	POST-TRAINING PRUNING
PROS	More efficient models since they are trained with the objective of sparsity in mind	Simpler to implement
	Pruning decisions are made alongside the consideration of model parameter optimization	Pruning parameters can be easily adjusted based on inference requirements
CONS	Makes training process more complex	May require fine-tuning to regain performance in the case of accuracy degradation
	Change in pruning parameters may require retraining of the whole model	Pruning decisions are user-defined and may not be optimal

TRADEOFFS OF DIFFERENT PRUNING APPROACHES

Structured vs Unstructured



Structured vs Unstructured

Input Operands	Accumulator	Dense TOPS	vs. FFMA	Sparse TOPS	vs. FFMA
FP32	FP32	19.5	–	–	–
TF32	FP32	156	8X	**312**	**16X**
FP16	FP32	312	16X	**624**	**32X**
BF16	FP32	312	16X	**624**	**32X**
FP16	FP16	312	16X	**624**	**32X**
INT8	INT32	624	32X	**1248**	**64X**

Table 1. Performance of Sparse Tensor Cores in the NVIDIA Ampere Architecture.

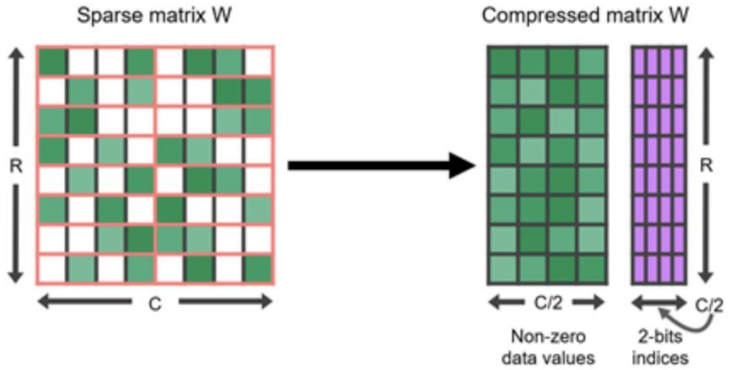
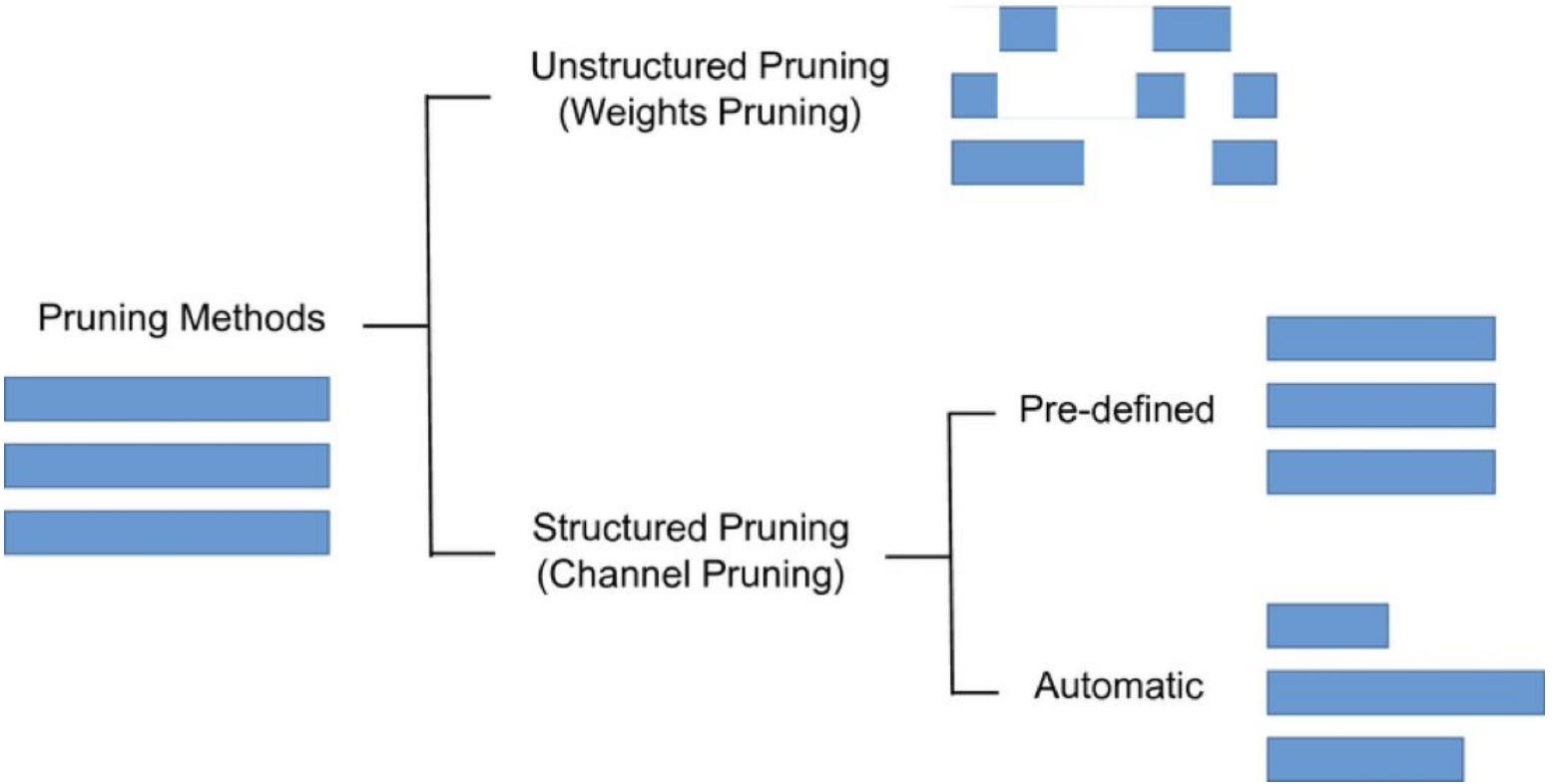
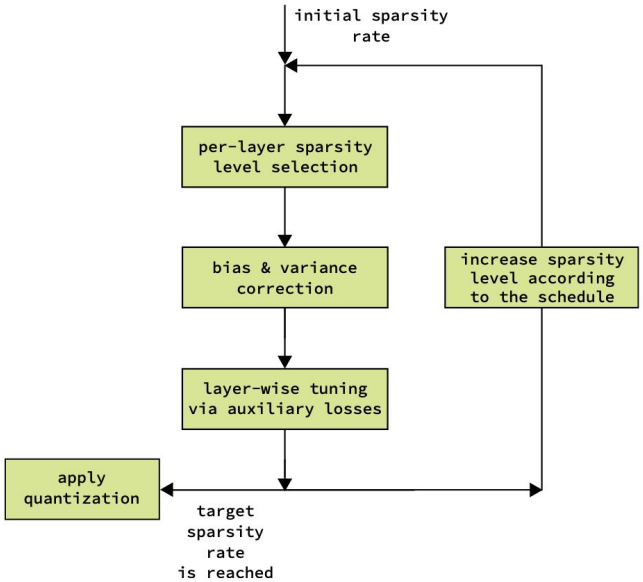


Figure 1. A 2:4 structured sparse matrix W, and its compressed representation

Structured vs Unstructured



Unstructured Pruning



$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t}{T}\right)^3$$

$$I(w_i^l) = \frac{|w_i^l|}{\sqrt{\sum_{j \in l} |w_j^l|^2}}$$

$$W_{corr}^s = \lambda W^s + E(W_{dense}) - E(\lambda W^s)$$

$$\lambda = \frac{\sigma(W_{dense})}{\sigma(W^s) + \epsilon}$$

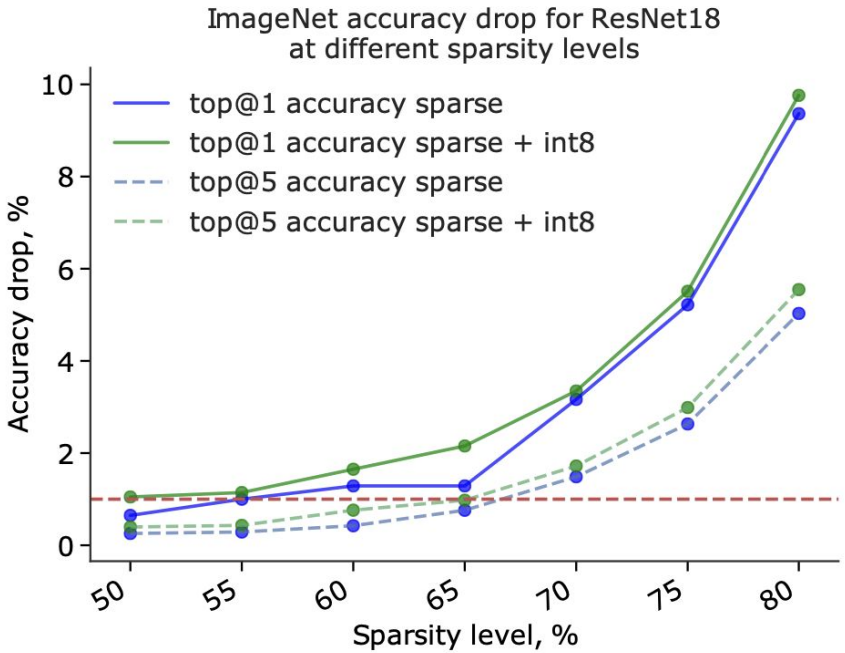
$$L = \sum_{i \in batch} (Y_{dense}^i - f(W^s M^s, X_{dense}^i) - b^s)^2$$

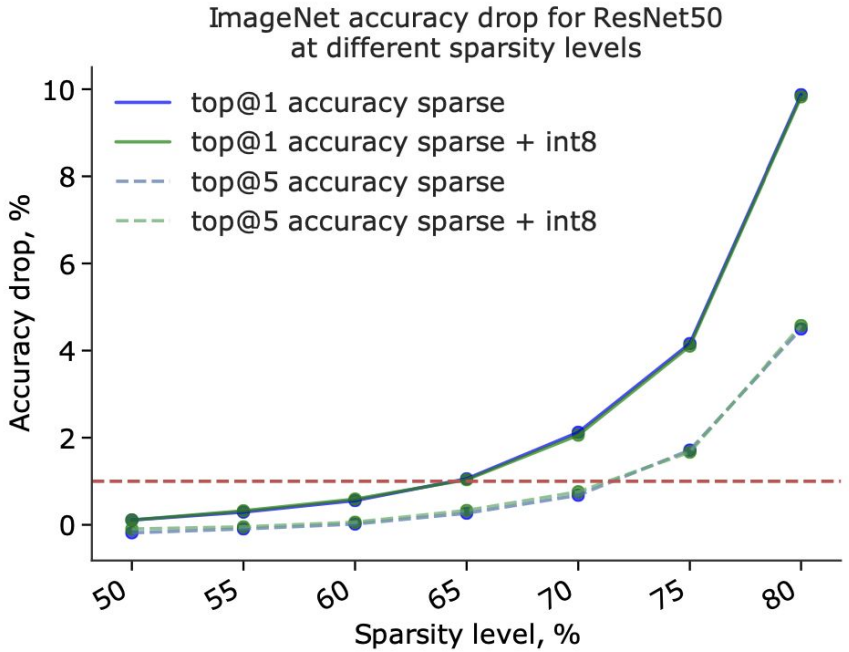
$$Y_{dense}^i = f(W_{dense}, X_{dense}^i) \quad (6)$$

$$b_{corr} = b_{dense} + E(f(W_{dense}, X_{dense})) - E(f(W_{corr}^s, X_{dense})) \quad (5)$$

Post-training deep neural network pruning via layer-wise calibration, Lazarevich et al, ICCV 2021

Unstructured Pruning





Post-training deep neural network pruning via layer-wise calibration, Lazarevich et al, ICCV 2021

Unstructured Pruning

Table 1. Accuracy values of the sparse 8-bit quantized DNN models obtained with the proposed post-training method. Metric values were measured on a CPU. The same is for other accuracy values reported in the paper unless specified otherwise.

Model	Dataset (acc. metric)	Sparsity rate, %	Compressed model acc.	Absolute acc. drop
ResNet50	ImageNet (top@1 acc.)	65	75.09	1.04
ResNet18	ImageNet (top@1 acc.)	50	68.93	0.81
GoogleNetV4	ImageNet (top@1 acc.)	50	78.96	0.94
MobileNetV2	ImageNet (top@1 acc.)	40	70.29	1.51
MobileNetV1-SSD	VOC07 (mAP)	50	71.53	0.98
TinyYOLOv2	COCO (AP)	50	28.29	0.83
NCF	MovieLens 20M (hit ratio)	70	64.67	0.93
BERT-base	MRPC (acc.)	50	82.50	0.63

Post-training deep neural network pruning via layer-wise calibration, Lazarevich et al, ICCV 2021

Structured Pruning of LLMs

While previous methods have effectively maintained model performance amidst parameter reduction, they primarily target compression within specialized domains or for designated tasks in the context of task-specific compression. Although this paradigm could potentially be employed for LLM compression, it compromises the LLM's capacity as a versatile task solver, rendering it suited to a single task exclusively. Thus, we strive to compress the LLM in a new setting: to reduce the LLM's size while preserving its diverse capabilities as general-purpose task solvers

- **The size of the training corpus of the LLM is enormous.** Previous compression methods heavily depend on the training corpus. The LLM has escalated the corpus scale to 1 trillion tokens or more [17, 51]. The extensive storage needs and protracted transmission times make the dataset difficult to acquire. Furthermore, if the dataset is proprietary, acquisition of the training corpus verges on impossibility, a situation encountered in [74, 39].
- **The unacceptably long duration for the post-training of the pruned LLM.** Existing methods require a substantial amount of time for post-training the smaller model [55, 28]. For instance, the general distillation in TinyBERT takes around 14 GPU days [20]. Even post-training a task-specific compressed model of BERT demands around 33 hours [61, 22]. As the size of both the model and corpus for LLMs increases rapidly, this step will invariably consume an even more extensive time.

LLM-pruner: On the Structural Pruning of Large Language Models, Xinyin Ma et al, NeurIPS 2023

Structured Pruning of LLMs

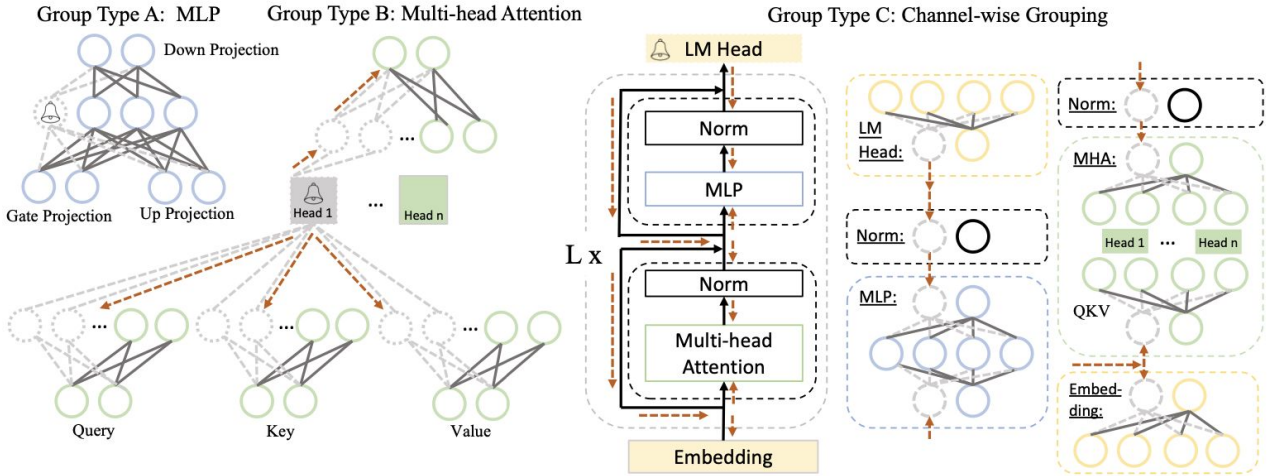


Figure 2: Illustration of the coupled structures in LLaMA. We simplify the neurons in each layer to make the dependent group clear. The trigger neuron, marked as a circle with a bell, cause weights with dependency pruned (dashed lines), which may propagate (red dashed lines) to coupled neurons (dashed circles). A group can be triggered by a variety of trigger neurons. Taking Group Type B as an example, the trigger for this group involves (i) the attention head, (ii) the output neuron in Query, Key or Value, and (iii) the input neuron in the final output projection.

LLM-pruner: On the Structural Pruning of Large Language Models, Xinyin Ma et al, NeurIPS 2023

Structured Pruning of LLMs

Vector-wise Importance. Suppose that given a dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$, where N is the number of samples. In our experiments, we set N equal to 10 and we use some public datasets as the source of $\mathcal{D}$. A group (as previously defined as a set of coupled structures) can be defined as $\mathcal{G} = \{W_i\}_{i=1}^{M}$, where M is the number of coupled structures in one group and W_i is the weight for each structure. While pruning, our goal is to remove the group that has the least impact on the model's prediction, which can be indicated by the deviation in the loss. Specially, to estimate the importance of W_i, the change in loss can be formulated as [24]:

$$I_{W_i} = |\Delta\mathcal{L}(\mathcal{D})| = |\mathcal{L}_{W_i}(\mathcal{D}) - \mathcal{L}_{W_i=0}(\mathcal{D})| = |\underbrace{\frac{\partial\mathcal{L}^{\top}(\mathcal{D})}{\partial W_i}W_i}_{\neq 0} - \frac{1}{2}{W_i}^{\top}HW_i + \mathcal{O}\left(\|W_i\|^3\right)| \quad (3)$$

LLM-pruner: On the Structural Pruning of Large Language Models, Xinyin Ma et al, NeurIPS 2023

Structured Pruning of LLMs

Element-wise Importance. The above can be considered as an estimate for the weight W_i. We can derive another measure of importance at a finer granularity, where each parameter within W_i is assessed for its significance:

$$I_{W_i^k} = |\Delta\mathcal{L}(\mathcal{D})| = |\mathcal{L}_{W_i^k}(\mathcal{D}) - \mathcal{L}_{W_i^k=0}(\mathcal{D})| = |\frac{\partial\mathcal{L}(\mathcal{D})}{\partial W_i^k} W_i^k - \frac{1}{2} W_i^k H_{kk} W_i^k + \mathcal{O}\left(\|W_i^k\|^3\right)| \quad (4)$$

Here, k represents the k-th parameter in W_i. The diagonal of the hessian H_{kk} can be approximated by the Fisher information matrix, and the importance can be defined as:

$$I_{W_i^k} = |\mathcal{L}_{W_i^k}(\mathcal{D}) - \mathcal{L}_{W_i^k=0}(\mathcal{D})| \approx |\frac{\partial\mathcal{L}(\mathcal{D})}{\partial W_i^k} W_i^k - \frac{1}{2}\sum_{j=1}^{N}\left(\frac{\partial\mathcal{L}(\mathcal{D}_j)}{\partial W_i^k} W_i^k\right)^2 + \mathcal{O}\left(\|W_i^k\|^3\right)| \quad (5)$$

LLM-pruner: On the Structural Pruning of Large Language Models, Xinyin Ma et al, NeurIPS 2023

Structured Pruning of LLMs

Group Importance. By utilizing either $I_{W_i^k}$ or I_{W_i}, we estimate the importance at the granularity of either a parameter or a vector of weight. Remembering that our goal is to estimate the importance of $\mathcal{G}$, we aggregate the importance scores in four ways: (i) Summation: $I_{\mathcal{G}} = \sum_{i=1}^{M} I_{W_i}$ or $I_{\mathcal{G}} = \sum_{i=1}^{M} \sum_k I_{W_i^k}$, (ii) Production: $I_{\mathcal{G}} = \prod_{i=1}^{M} I_{W_i}$ or $I_{\mathcal{G}} = \prod_{i=1}^{M} \sum_k I_{W_i^k}$, (iii) Max: $I_{\mathcal{G}} = \max_{i=1}^{M} I_{W_i}$ or $I_{\mathcal{G}} = \max_{i=1}^{M} \sum_k I_{W_i^k}$; (iv) Last-Only: Since deleting the last executing structure in a dependency group is equivalent to erasing all the computed results within that group, we assign the importance of the last executing structure as the importance of the group: $I_{\mathcal{G}} = I_{W_l}$ or $I_{\mathcal{G}} = \sum_k I_{W_l^k}$, where l is the last structure. After assessing the importance of each group, we rank the importance of each group and prune the groups with lower importance based on a predefined pruning ratio.

LLM-pruner: On the Structural Pruning of Large Language Models, Xinyin Ma et al, NeurIPS 2023

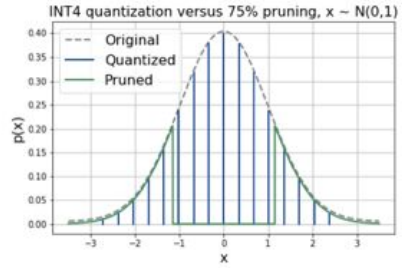
Structured Pruning of LLMs

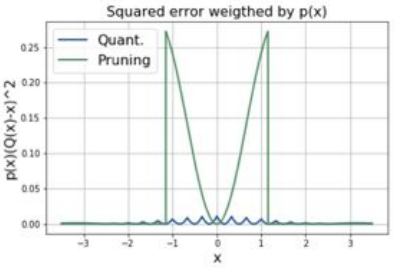
Table 1: Zero-shot performance of the compressed LLaMA-7B. The average is calculated among seven classification datasets. 'Underline' indicates the best pruning-only performance, while 'bold' represents the overall best performance with the same pruning ratio, considering both pruning and post-training. The 'Channel' strategy only prunes the dependent group of Type C, while all other methods employ the 'Block' strategy to prune dependent groups in both Type A and Type B. Since [51] did not provide its prompt, the evaluation of the result with * is performed under different prompts, which is lower than the official results.

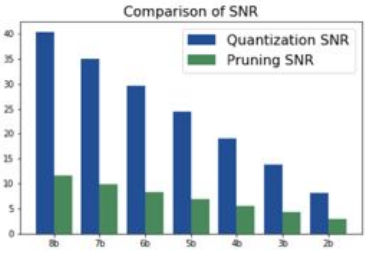
Pruning Ratio	Method	WikiText2↓	PTB↓	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
Ratio = 0%	LLaMA-7B[51]	-	-	76.5	79.8	76.1	70.1	72.8	47.6	57.2	68.59
	LLaMA-7B*	12.62	22.14	73.18	78.35	72.99	67.01	67.45	41.38	42.40	63.25
Ratio = 20% w/o tune	L2	582.41	1022.17	59.66	58.00	37.04	52.41	33.12	28.58	29.80	42.65
	Random	27.51	43.19	61.83	71.33	56.26	54.46	57.07	32.85	35.00	52.69
	Channel	74.63	153.75	62.75	62.73	41.40	51.07	41.38	27.90	30.40	45.38
	Vector	22.28	41.78	61.44	71.71	57.27	54.22	55.77	33.96	38.40	53.25
	Element^2	19.77	36.66	59.39	75.57	65.34	61.33	59.18	37.12	39.80	56.82
	Element^1	19.09	34.21	57.06	75.68	66.80	59.83	60.94	36.52	**40.00**	56.69
Ratio = 20% w/ tune	Channel	22.02	38.67	59.08	73.39	64.02	60.54	57.95	35.58	38.40	55.57
	Vector	18.84	33.05	65.75	74.70	64.52	59.35	60.65	36.26	39.40	57.23
	Element^2	**17.37**	30.39	**69.54**	76.44	68.11	**65.11**	63.43	**37.88**	**40.00**	**60.07**
	Element^1	17.58	**30.11**	64.62	**77.20**	**68.80**	63.14	**64.31**	36.77	39.80	59.23

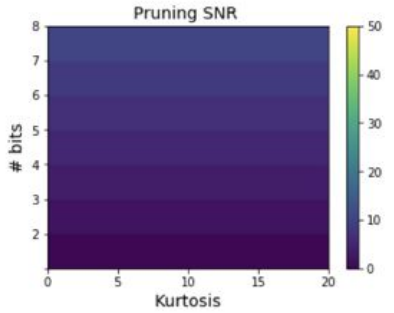
LLM-pruner: On the Structural Pruning of Large Language Models, Xinyin Ma et al, NeurIPS 2023

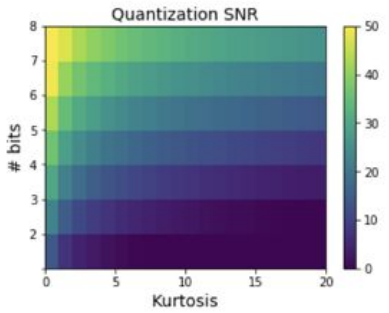
Quantization vs Pruning

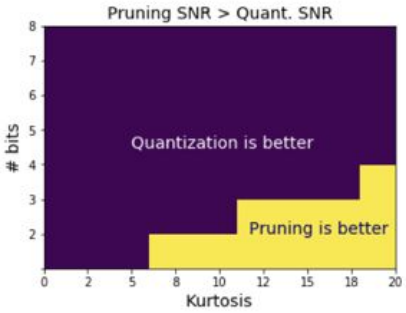












Pruning vs Quantization: Which is Better?, Kuzmin et al, Neurips, 2023

Quantization vs Pruning

$$\min_{\boldsymbol{w}} E(\boldsymbol{w}) = \|\boldsymbol{X}\delta\boldsymbol{w} - \boldsymbol{X}\boldsymbol{w}_{orig}\|_2^2$$
$$\text{s.t. } \boldsymbol{w} \in \mathbb{Z}^n,$$
$$w_{min} \leq w_i \leq w_{max},$$

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{P}\boldsymbol{w} - \boldsymbol{q}^T\boldsymbol{w},$$
$$\text{s.t. } \boldsymbol{w} \in \mathbb{Z}^n,$$
$$w_{min} \leq w_i \leq w_{max},$$

$$L(\boldsymbol{\lambda}) = \max -\gamma,$$
$$\text{s.t. } \begin{bmatrix} \boldsymbol{P} - \mathbf{diag}(\boldsymbol{\lambda}) & q + \frac{1}{2}\lambda \\ \left(q + \frac{1}{2}\lambda\right)^T & \gamma \end{bmatrix} \succeq 0,$$
$$\boldsymbol{\lambda} \geq 0,$$

$$E = \min_{\hat{\boldsymbol{w}}} \|\boldsymbol{X}\hat{\boldsymbol{w}} - \boldsymbol{X}\boldsymbol{w}_{orig}\|_2^2$$
$$\text{s.t. } \|\hat{\boldsymbol{w}}\|_0 \leq s,$$

$$E(\boldsymbol{w}) = \min_{\boldsymbol{w},\boldsymbol{m}} \|\boldsymbol{X}(\boldsymbol{m} \odot \boldsymbol{w}) - \boldsymbol{X}\boldsymbol{w}_{orig}\|_2^2,$$
$$\text{s.t. } \|\boldsymbol{m}\|_1 = s,$$
$$-\boldsymbol{m} \odot l \leq \hat{\boldsymbol{w}} \leq \boldsymbol{m} \odot u$$
$$l, u > 0, m_i \in \{0, 1\},$$

Pruning vs Quantization: Which is Better?, Kuzmin et al, Neurips, 2023

Quantization vs Pruning

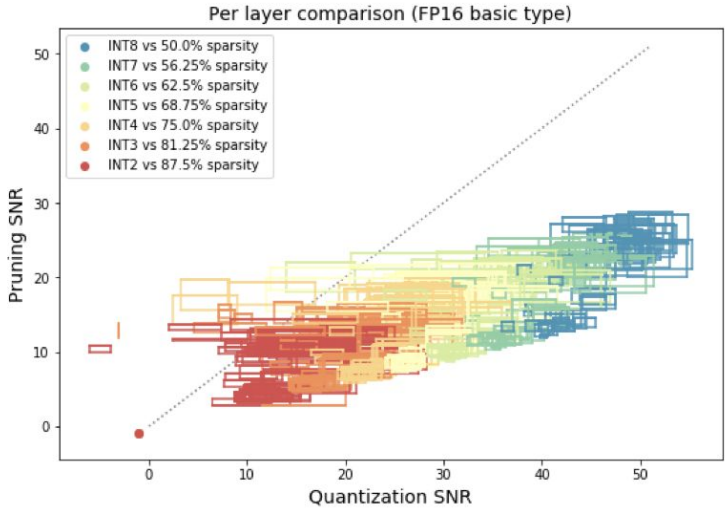


Figure 4: Comparison in the post-training scenario. Each box corresponds to a subset of one of 10 layers from the 4 different models that were used, with 7 different bit-width comparison points. The ranges of the box indicate the lower and higher-bounds found by the algorithms.

Pruning vs Quantization: Which is Better?, Kuzmin et al, Neurips, 2023

Quantization vs Pruning

Model	Orig.	Metric	Method	8b	7b	6b	5b	4b	3b	2b
Resnet-18	69.7	acc.	quant.	**70.5**	**70.5**	**70.6**	**70.3**	**70.0**	**68.9**	**67.3**
			pruning	70.3	70.1	69.9	69.5	69.3	68.3	66.8
Resnet-50	76.1	acc.	quant.	76.4	**76.4**	**76.4**	**76.3**	**76.2**	**75.5**	72.3
			pruning	**76.6**	**76.4**	76.2	76.1	75.9	75.4	**74.3**
MobileNet-V2	71.7	acc.	quant.	**71.9**	**72.0**	**71.7**	**71.6**	**70.9**	**68.6**	**59.1**
			pruning	68.1	65.6	61.9	56.3	48.0	34.0	21.2
EfficientNet	75.4	acc.	quant.	**75.2**	**75.3**	**75.0**	**74.6**	**74.0**	**71.5**	**60.9**
			pruning	72.5	70.9	68.1	63.6	56.4	44.5	27.1
MobileNet-V3	67.4	acc.	quant.	**67.7**	**67.6**	**67.1**	**66.3**	**64.7**	**60.8**	**50.5**
			pruning	65.6	64.4	62.4	60.2	56.1	31.7	0.0
ViT	81.3	acc.	quant.	**81.5**	**81.4**	**81.4**	**81.0**	**80.4**	**78.4**	**72.2**
			pruning	76.6	76.6	76.2	73.1	72.4	71.5	69.4
DeepLab-V3	72.9	mIoU	quant.	**72.3**	**72.3**	**72.4**	**71.9**	**70.8**	**63.2**	**17.6**
			pruning	65.2	62.8	56.8	47.7	32.9	18.6	10.0
EfficientDet	40.2	mAP	quant.	**39.6**	**39.6**	**39.6**	**39.2**	**37.8**	**33.5**	**15.5**
			pruning	34.5	33.0	30.9	27.9	24.2	17.9	8.0
OPT-350m	14.8	perpl.	quant.	**14.8**	**14.8**	**14.9**	**15.0**	**15.3**	**15.9**	**19.9**
			pruning	18.0	19.7	22.6	27.2	35.4	53.5	101.4

Table 1: Comparison of QAT and magnitude pruning with fine-tuning given equal model size and equal number of epochs for fine-tuning.

Pruning vs Quantization: Which is Better?, Kuzmin et al, Neurips, 2023

Quantization vs Pruning

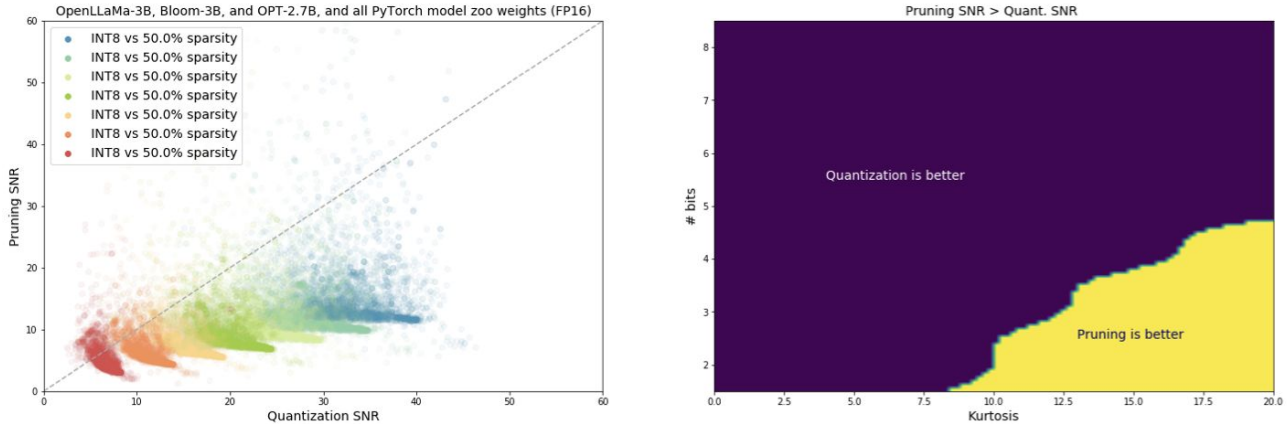


Figure 3: (left) Comparison on all the weights from PyTorch model zoo (46 models) combined with 3 large language models (Bloom-3b, Llama-3b, OPT-2.7b). (left) Pruning SNR versus quantization SNR for every tensor. (right) Pruning is preferable at high compression ratios for tensors with high sample kurtosis values.

Pruning vs Quantization: Which is Better?, Kuzmin et al, Neurips, 2023

Если понравилась лекция

Aligning Distributionally Robust Optimization with Practical Deep Learning Needs, Feoktistov et al, arxiv preprint, 2025