BBQ: Boosting Quantization Entropy with Bell Box Quantization

Anonymous authors
Paper under double-blind review

Abstract

Quantization-Aware Pre-Training (QAPT) is an effective technique to reduce the compute and memory overhead of Deep Neural Networks while improving their energy efficiency on edge devices. Existing QAPT methods produce models stored in compute-efficient data types (e.g. integers) that are not information theoretically optimal (ITO). On the other hand, existing ITO data types (e.g. Quantile/NormalFloat Quantization) are not compute-efficient. We propose BBQ, the first ITO quantization method that is also compute-efficient. BBQ builds on our key insight that since learning is domain-agnostic, the output of a quantizer does not need to reside in the the same domain as its input. BBQ performs ITO quantization in its input domain, and returns its output in a compute-efficient domain where ITO data types are mapped to compute-efficient data types. Without sacrificing compute efficiency, BBQ outperforms prior SOTA QAPT methods by a perplexity reduction of up to 2 points for 4-bit models, up to 4 points for 3-bit models, up to 5 points for 2-bit models, and up to 18 points for 1-bit models.

1 Introduction

Quantization is an effective method to reduce the computation/memory/energy consumption of Deep Neural Networks (DNNs), allowing DNNs to be deployed to edge devices with limited hardware resources. However, quantization often degrades model quality. While Post-Training Quantization (PTQ) methods (Lin et al., 2024; Shao et al., 2024; Ma et al., 2024b; Liu et al., 2024a; Kim et al., 2024) can mitigate quality degradation without re-training for higher precisions (4+-bit), they struggle to maintain quality when weights and activations are quantized to 4-bit and below (Panferov et al., 2025; Esser et al., 2020; Ma et al., 2024a; Kumar et al., 2025). Quantization-Aware Training (QAT) methods (Panferov et al., 2025; Esser et al., 2020; Ma et al., 2024a; Kumar et al., 2025; Shen et al., 2024; Liu et al., 2025; Yamamoto, 2021), on the other hand, can achieve higher accuracy than PTQ methods under the same precision (Du et al., 2024; Panferov et al., 2025; Liu et al., 2024b) by introducing quantization in the training loop.

QAT can be further divided into Quantization-Aware Pre-Training (Panferov et al., 2025; Shen et al., 2024) (QAPT) and Quantization-Aware Fine-Tuning (Malinovskii et al., 2024; Du et al., 2024) (QAFT). QAFT initializes a low-precision model from a full-precision pre-trained checkpoint, and trains the model for a short duration to fit a downstream task, which is typically much smaller than the pre-training dataset (Du et al., 2024). On the other hand, QAPT initializes a low-precision model from scratch, aims to fit a much larger dataset, and typically trains for much longer durations. Compared to first pre-train in full-precision and subsequently apply PTQ/QAFT, QAPT may have higher pre-training speed (Kumar et al., 2025; Xi et al., 2023; Castro et al., 2025), as when activations and weights are quantized, the forward pass of training may be performed in low precision.

This work aims to improve the accuracy of a QAPT'ed model without compromising its efficiency on edge devices. Specifically, we target the case when a low-precision model is initialized randomly, trained for long durations on large datasets, and is aimed to be deployed to edge devices with constraints on memory capacity, inference latency, and energy consumption. The energy constraint requires the model to be compute-efficient, or that expensive high-precision matrix multiplications be substituted with low-precision arithmetic, while the memory and latency constraints require the model to be small in size, or to have a low parameter count and precision. The main challenge of

Signed INT4	-8, ±7, ±6, ±5, ±4, ±3, ±2, ±1, 0
MX FP4	±6, ±4, ±3, ±2, ±1.5, ±1, ±0.5, ±0

Table 1: Possible values of the INT4 and the MX FP4 data type.

this work is models with limited memory footprint cannot fit large datasets well due to a lack of learning capacity (Kumar et al., 2025).

Using the discrete Shannon entropy (Shannon, 1948; Cover & Thomas, 2006) of quantized weights as a proxy to the amount of information/knowledge present in a QAPT'ed model, we observe that SOTA QAPT methods QuEST (Panferov et al., 2025) and LSQ (Esser et al., 2020) produce quantized models that under-utilize the available learning capacity. The under-utilization is because LSQ and QuEST use compute-efficient data types (e.g. integers and floats) which are not information theoretically optimal (ITO). While existing ITO data types (Dettmers et al., 2023; 2022) can reduce this under-utilization of learning capacity, they lack compute efficiency on modern CPUs/GPUs, which limits the applicability on edge devices with limited energy.

To maximally utilize the limited learning capacity while preserving compute efficiency, we propose Bell Box Quantization (BBQ). BBQ is designed based on our key insight: since learning is domain-agnostic, the output of a quantizer does not need to live in the same domain as its input. BBQ performs ITO quantization in its input domain to maximally preserve information, but returns its output in a different domain, where ITO data types are mapped to compute-efficient data types. BBQ achieves higher capacity utilization and better prediction quality than QuEST and LSQ. Unlike existing ITO data types, a BBQ-quantized model can still accelerate expensive matrix multiplications with low-precision arithmetic.

2 Background and Motivation

In this section, we discuss compute-efficient data types on modern GPUs, existing ITO quantization methods, and lastly, the domain-agnostic property of learning which is the key inspiration of BBQ.

Quantization is the discretization of a continuous random variable x to a discrete random variable $\hat{x}$, with 2^b possible states, such that $\hat{x}$ is sufficiently close to x. We present a generic template for quantization as follows:

$$\hat{x} = f^{-1}(q), \text{ where } q = r(f(x)) \tag{1}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a transform function that converts x to some intermediate domain more suitable for quantization; $r : \mathbb{R} \to \mathbb{T}$ is some rounding function that converts any real value to one of 2^b possible real values, i.e. $\mathbb{T} \subseteq \mathbb{R}$ and $|\mathbb{T}| = 2^b$; and f^{-1}, the inverse function of f, is responsible for converting the discretized value q back to the original domain of x.

2.1 Compute Efficient Data Types

A data type $\mathbb{T}$ is compute-efficient, if hardware offers low-precision multiply-accumulate instructions that directly operate on members of $\mathbb{T}$ without first decoding to high precision data types. Table 1 shows members of $\mathbb{T}$ for INT4, which is supported by the Nvidia Ampere (Nvidia, 2021) and Turing (Nvidia, 2018) architectures, and members of $\mathbb{T}$ for MX FP4 (Rouhani et al., 2023), which is supported by the Blackwell architecture (Nvidia, 2025). When f^{-1} is linear or affine, a compute-efficient data type can be used to accelerate matrix multiplication and convolution (Yao et al., 2021). For example, if $f^{-1}(q) = sq$ for a constant scalar s, then the multiplication of activation matrix $\hat{X} = f^{-1}(Q_x) = sQ_x$ and weight matrix $\hat{W} = f^{-1}(Q_w) = sQ_w$ can be simplified to $s^2(Q_x \cdot Q_w)$. In other words, the matrix multiplication can be computed using solely low-precision representations of activations and weights, and later de-quantized to the original domain by multiplication of s^2.

Prior SOTA QAPT methods QuEST (Panferov et al., 2025) and LSQ (Esser et al., 2020) uses linear/affine f^{-1} and compute-efficient data types. We show the definition of f, r, and f^{-1} for LSQ and QuEST as follows:

$$\text{LSQ: } f(x) = x/s, r(v) = \lfloor \text{clip}(v, -2^{b-1}, 2^{b-1} - 1) \rceil, f^{-1}(q) = sq$$
$$\text{QuEST: } f(x) = \frac{\text{HT}(x)}{\alpha^* \sigma} - \frac{1}{2}, r(v) = \lfloor \text{clip}(v, -2^{b-1}, 2^{b-1} - 1) \rceil, f^{-1}(q) = \text{HT}(\alpha^* \sigma (q + \frac{1}{2})) \tag{2}$$

where s, σ, and α^* are constant scalars; HT is the Hadamard transform, a linear operation; and the clip operation and the round-to-nearest-integer operation $\lfloor \cdot \rceil$ enforces the output of r to be a b-bit signed integer. Since f^{-1} is linear or affine in LSQ and QuEST, matrix multiplications can be accelerated with low-precision arithmetic.

In short, for a quantization method to be compute-efficient, it should have a linear/affine dequantization function f^{-1} and a compute-efficient data type $\mathbb{T}$.

2.2 Information Theoretically Optimal Quantization Methods

An ITO quantization method is one that ensures each member of $\mathbb{T}$ is used equally often (Dettmers et al., 2023). For example, if $b = 2$, then 25% of values of x should be assigned to each of the 4 possible values in $\mathbb{T}$. While there are many possible mappings that evenly split the probability mass, a trivial mapping is to use the 25, 50, 75 percentile as rounding boundaries. In other words, values of x less than the 25-percentile of x are rounded to the first value in $\mathbb{T}$; values larger than the 25-percentile but less than 50-percentile of x are rounded to the second value, etc. For $x \sim N(0, \sigma^2)$, this trivial mapping can be expressed as:

$$f(x) = x/\sigma, r(v) = T[\text{floor}(2^b \Phi(v))], f^{-1}(q) = \sigma q \tag{3}$$

where $T[i]$ is the ith smallest element in $\mathbb{T}$. Note that Equation 3 only requires that $\mathbb{T}$ contains 2^b unique real values. Therefore, Equation 3 describes a set of ITO methods rather than a unique one. By evenly splitting the probability mass of x, all ITO data types maximize the Shannon entropy of q defined as

$$H(q) = \sum_{t \in \mathbb{T}} -P(q = t) \log_2 P(q = t) \tag{4}$$

Note that entropy was shown to positively correlate with accuracy/prediction quality (Cheng et al., 2025; Shen et al., 2024).

Inspired by ITO quantization methods, Dettmers et al. (2023) proposed NormalFloat, whose values are a list of abs-max-normalized Gaussian quantiles (Yoshida, 2023; Dotzel et al., 2024). Due to lack of hardware support, NormalFloat must be de-quantized to full-precision floats and then used in computation, which limits its applicability on energy-constrained edge devices.

This presents a dilemma: while ITO quantization methods maximally preserve information, they are not compute-efficient; compute-efficient quantization methods, on the other hand, are not ITO for Gaussianly distributed weights/activations. Can we obtain the best of both worlds?

2.3 Learning is Domain-Agnostic

Since neural networks are universal function approximators (Hornik et al., 1989), they are capable of learning from transformed/augmented data, or even data encoded in latent spaces that are not human-understandable. For example, DNNs can correctly classify images that are rotated (He et al., 2015), or even images that are transformed to the frequency domain (Xu et al., 2020). As another example, when sparse autoencoders (Gao et al., 2025) are trained just to reconstruct the data, the resulting latent embedding of the data can be used by other DNNs to complete downstream tasks. These are all evidence that, as long as information is preserved, simply projecting/transforming data to a different domain does not prevent learning.

The domain-agnostic property of learning provides an opportunity to circumvent the inefficiency of ITO quantization methods. Rather than treating quantizers as data compressors and reconstructors, we could treat them as feature extractors that return a set of compact latent features of the original data. Critically, these latent features reside in an output domain, that is, not necessarily the same domain as the input, but is designed to be a compute-efficient domain, where features can be efficiently used by matrix multiplication operators. In other words, we ask the research question: *if we*

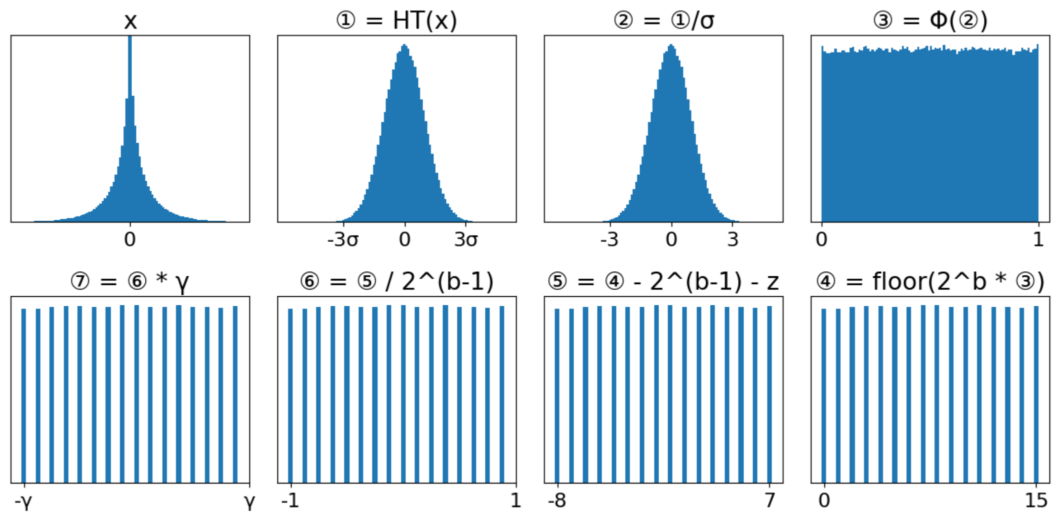

Figure 1: The seven steps of BBQ: Hadamard Transform (①), RMS normalization (②), probability integral transform (③), uniform quantization (④), unsigned-to-signed conversion (⑤), precision-dependent scaling (⑥), and precision-independent scaling (⑦).

design a quantizer that performs ITO quantization in the original domain, and returns output in an alternative compute-efficient domain, will such a quantizer outperform existing non-ITO quantizers in QAPT?

3 Proposed Method

In this section, we present the Bell Box Quantizer (BBQ). BBQ is designed to be ITO, or to maximally preserve information from its input, while also returning compute-efficient outputs that can be accelerated by modern hardware. The formulation of BBQ is as follows:

$$\begin{aligned} q &= \lfloor 2^b \Phi\left(\mathrm{HT}(x)/\sigma\right) \rfloor - 2^{b-1} - z \\ \hat{x} &= \frac{\gamma}{2^{b-1}} q \end{aligned} \tag{5}$$

where γ is a learnable scaling factor, HT is the Hadamard transform operation, Φ is the Gaussian CDF, $\lfloor \cdot \rfloor$ is the floor operation, σ is the RMS normalization factor $\sqrt{E[(\mathrm{HT}(x))^2]}$, and z is a hyperparameter zero point. As illustrated in Figure 1, BBQ involves seven steps: Hadamard Transform, RMS Normalization, probability integral transform, uniform quantization, unsigned-to-signed conversion, precision-dependent scaling, and precision-independent scaling.

3.1 Step 1: Hadamard Transform

Similar to QuEST (Panferov et al., 2025), the first step of BBQ is to Gaussianize the input x by performing the Hadamard transform, visualized in Figure 1 Row 1 Column 2. After the transform, $\mathrm{HT}(x)$ behaves like samples from $N(0, \sigma^2)$. Instead of transforming the whole matrix x, we follow QuEST and perform the Hadamard Transform on every H elements of x along the input channel dimension. During training, the Hadamard Transform is a differentiable operation, and we simply use autograd frameworks to perform its backward pass. During inference, the Hadamard transform is implemented with a vector-matrix multiplication, between an H-element slice of x and the pre-computed $H \times H$ Hadamard matrix.

3.2 Step 2: RMS Normalization

The next step of BBQ, visualized in Figure 1 Row 1 Column 3, is to normalize the Gaussian-like data by dividing the root-mean-square scaling factor $\sigma = \sqrt{E[\mathrm{HT}(x)]}$. After the normalization,

Algorithm 1 Example 3-bit Inference Quantization Kernel for a Thread Block

```
 1: function BBQ(block_id, x_ptr, h_ptr, q_ptr, $E[1/\sigma]$)
 2:     x = x_ptr[block_id:block_id+1, :] # x_ptr is an $N \times N$ matrix reshaped to $(N^2/H, H)$
 3:     h = h_ptr[:, :] # h_ptr is a pre-computed Hadamard matrix with shape $(H, H)$
 4:     x = x @ h # vector matrix multiplication between shapes $(1, H)$ and $(H, H)$
 5:     x = x * $E[1/\sigma]$ # element-wise multiplication by scalar
 6:     for $t \leftarrow 0$ to $H - 1$ do # can be done in parallel across warps and threads
 7:         if $x[t] \geq \Phi^{-1}(4/8)$ then # Binary search through pre-computed $\Phi^{-1}$ values
 8:             if $x[t] \geq \Phi^{-1}(6/8)$ then
 9:                 x[t] = 3 if $x[t] \geq \Phi^{-1}(7/8)$ else 2
10:             else
11:                 x[t] = 1 if $x[t] \geq \Phi^{-1}(5/8)$ else 0
12:             end if
13:         else
14:             if $x[t] \geq \Phi^{-1}(2/8)$ then
15:                 x[t] = -1 if $x[t] \geq \Phi^{-1}(3/8)$ else -2
16:             else
17:                 x[t] = -3 if $x[t] \geq \Phi^{-1}(1/8)$ else -4
18:             end if
19:         end if
20:     end for
21:     q_ptr[id:id+1, :] = x # store quantized results to global memory
22: end function
```

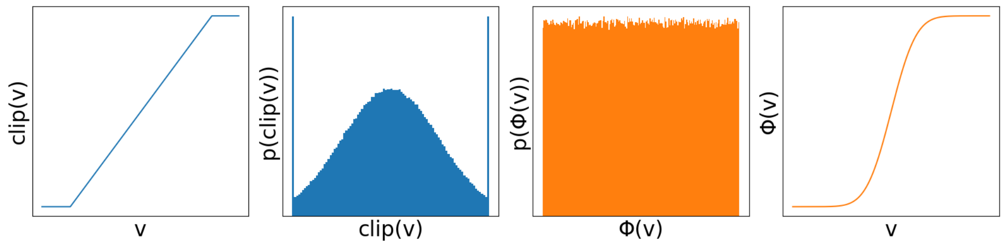

Figure 2: Comparison of clipping (blue) and the normal CDF (orange) Φ for $v \sim N(0, 1)$.

the data behaves like samples from $N(0, 1)$. For weight quantization during training, we measure σ and perform RMS normalization for every output channel individually. After training, weight quantization can be done once offline and does not introduce runtime overhead. For activation quantization, we measure σ and perform RMS normalization for the whole activation tensor. In addition, during training we keep track of an exponential moving average of $1/\sigma$ measured from activations, denoted as $E[1/\sigma]$. During inference, instead of measuring $1/\sigma$ from activation tensors, we use the recorded $E[1/\sigma]$ instead. This is similar to how BatchNorm (Ioffe & Szegedy, 2015) uses running statistics during inference. During training, we use autograd frameworks to perform the backward pass of RMS Normalization to x and σ. Unlike QuEST, we do not detach the gradient of σ.

3.3 Step 3: Probability Integral Transform

The probability integral transform (Fisher, 1992) converts any continuous distribution to a uniform one by applying its CDF on the data. As visualized in Figure 1 Row 1 Column 4, we apply the standard Gaussian CDF Φ to data that behaves like samples from $N(0, 1)$, creating data that behaves likes samples from $U(0, 1)$. In QuEST and LSQ, the component that behaves similarly to Φ is the clip operation. As shown in Figure 2, both functions restrict their output to be within some finite range. However, Φ is infinitely differentiable and is therefore much smoother than clip which is piecewise linear. A smoother operation can be more suitable for optimization methods like Gradient Descent (Nesterov, 2014; Bottou et al., 2018). For example, the GELU function (Hendrycks & Gimpel, 2023) defined as GELU$(x) = x\Phi(x)$ is smoother than the piecewise linear ReLU (Agarap,

Precision	Possible Values of q
4	-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7
3	-4,-3,-2,-1,0,1,2,3
2	-1.5,-0.5,0.5,1.5
1	-0.5,0.5

Table 2: Values that BBQ can represent. As shown, for $b = 4$, BBQ can be encoded as INT4. For $b = 3$, BBQ can be encoded as INT4 or FP4. For $b \in \{1, 2\}$, BBQ can be encoded as FP4.

2018), and GELU can outperform ReLU empirically (Hendrycks & Gimpel, 2023). During training, Φ is a differentiable operation and we use the autograd framework to perform BackProp. During inference, we combine Φ with the subsequent floor operation using the following property:

$$\lfloor 2^b \Phi(v) \rfloor = i, \text{ s.t. } \Phi^{-1}\left(\frac{i}{2^b}\right) \leq v < \Phi^{-1}\left(\frac{i+1}{2^b}\right) \tag{6}$$

which allows us to pre-compute $\Phi^{-1}\left(\frac{i}{2^b}\right)$ for all 2^b possible values of i, and at runtime perform a binary search that requires b floating point comparisons. The pseudocode of a 3-bit inference kernel implementation of BBQ is shown in Algorithm 1.

3.4 Step 4 and 5: Uniform Quantization and Unsigned-to-Signed Conversion

At this stage, uniform quantization is the ITO quantization method since the data behaves like samples from $U(0, 1)$ as shown in Figure 1 Row 1 Column 4. Since it is known that having zero-mean activations is desired for training DNNs (LeCun et al., 2012), we subsequently convert the positive-only data to symmetric data by subtracting 2^{b-1} and the hyperparameter zero point z, producing the final q stored as a compute-efficient data type, visualized in Figure 1 Row 2 Column 3. Inspired by NF4 (Dettmers et al., 2023), we use $z = 0$ for $b \in \{3, 4\}$, allowing the data type to represent zero exactly while sacrificing an extra value on the positive side. We use $z = -0.5$ for $b \in \{1, 2\}$ to ensure activations remain zero-mean. During training, we use the Straight-Through Estimator (Bengio et al., 2013) for the floor operation, and the autograd framework for all other differentiable operations. Table 2 lists the values supported by the BBQ data type. For $b \in \{1, 2, 3\}$, BBQ can be represented using the MX FP4 data type on existing Blackwell GPUs. For $b \in \{3, 4\}$, BBQ can be represented using INT4 on existing Ampere and Turing GPUs.

3.5 Step 6 and 7: Precision Dependent and Independent Scaling

BBQ includes a learnable scaling parameter $s = \gamma/2^{b-1}$, as without it, the network has no control over the magnitude of its activations. Normalization layers like BatchNorm (Ioffe & Szegedy, 2015) and LayerNorm (Ba et al., 2016) use a similar learnable scaling factor to address the same problem. Instead of learning s like in LSQ (Esser et al., 2020), we decouple s into the ratio of a precision-independent learnable scaling factor γ (visualized in Figure 1 Row 2 Column 1) and a constant precision-dependent scaling factor 2^{b-1} (visualized in Figure 1 Row 2 Column 2) to ensure γ can be initialized to the same value regardless of precision.

The initialization of γ also plays a critical role, as overly large initialization can lead to gradient explosion while overly small initialization can lead to gradient vanishing. We initialize γ to $\zeta^* \sigma_0$, where σ_0 is the σ measured at the first iteration of training, and ζ^* is a constant MSE optimal scaling factor defined as follows:

$$\zeta^* = \arg\min_{\zeta} E_{v \sim N(0,1)}[(v - \zeta(2\Phi(v) - 1))^2] \tag{7}$$

In other words, if a Gaussian variable v is transformed to the cumulative distribution domain and scaled to the range $[-\zeta, \zeta]$, the resulting quantity $\zeta(2\Phi(v) - 1)$ is closest on average to v when $\zeta = \zeta^*$. By initializing γ to $\zeta^* \sigma_0$, $\hat{x}$ can have approximately the same magnitude as x (in the first iteration) to prevent the activation magnitude from exploding or vanishing as tokens travel deeper into the network. After the first iteration, we simply let γ update itself based on gradient descent. In addition, we apply an LSQ-style gradient scaling method (Esser et al., 2020) to reduce the gradient of γ by a factor of $\sqrt{n}$, where n is the number of weights/activations in tensor x. We use per-channel γ for weights, and per-tensor γ for activations. We do not apply weight decay to γ.

Params	Tokens	Full	Bits	BBQ		QuEST		LSQ	
				Entropy	Perplexity	Entropy	Perplexity	Entropy	Perplexity
95M	3B	**24.75**	4	**3.93**	**25.51**	3.61	26.37	3.59	27.46
95M	3B	**24.75**	3	**2.96**	**26.55**	2.78	29.04	2.74	30.27
95M	3B	**24.75**	2	**1.97**	**31.34**	1.92	35.58	1.69	36.58
95M	3B	**24.75**	1	**1.00**	**49.22**	**1.00**	67.78	-	-
125M	5B	**21.51**	4	**3.93**	**22.15**	3.61	22.98	3.60	23.77
125M	5B	**21.51**	3	**2.96**	**23.22**	2.78	25.21	2.74	26.28
125M	5B	**21.51**	2	**1.98**	**27.34**	1.93	31.32	1.81	31.42
125M	5B	**21.51**	1	**1.00**	**44.58**	**1.00**	72.54	-	-
200M	10B	**17.93**	4	**3.93**	**18.79**	3.61	19.06	2.73	1778
200M	10B	**17.93**	3	**2.96**	**19.74**	2.78	20.82	2.50	140.9
200M	10B	**17.93**	2	**1.98**	**23.08**	1.93	25.46	1.63	78.19
200M	10B	**17.93**	1	**1.00**	**38.27**	**1.00**	52.37	-	-
300M	20B	**15.43**	4	**3.93**	**16.10**	3.61	16.26	-	-
300M	20B	**15.43**	3	**2.96**	**16.90**	2.78	17.67	-	-
300M	20B	**15.43**	2	**1.98**	**19.75**	1.93	21.53	-	-

Table 3: Entropy and Perplexity of LLaMA models pretrained on the C4 dataset. The headers are the number of parameters (Params), the number of tokens (Tokens), perplexity without any quantization (Full), activation/weight precision (Bits), and the entropy and perplexity of BBQ, QuEST and LSQ.

Params	95M	95M	95M	95M	95M	95M	95M	95M	95M	95M	95M
Bits WA	16	4	4	4	4	3	3	3	3	1	1
Method	None	BBQ	QuEST	LSQ	NF4	BBQ	QuEST	LSQ	NF3	BBQ	QuEST
Perplexity	25	**26.7**	26.83	27.55	28.5	**28.61**	29.7	30.85	36.16	**53.8**	77.96
Params	95M	95M	95M	95M	95M	125M	125M	125M	125M	125M	125M
Bits WA	2	2	2	2	2	3	3	2	2	1	1
Method	BBQ	QuEST	LSQ	NF2	SEQ	BBQ	QuEST	BBQ	QuEST	BBQ	QuEST
Perplexity	**34.36**	36.7	38.98	246.6	40.07	**24.78**	26.05	**29.38**	31.47	**50.46**	69.78

Table 4: Perplexity of GPT models pretrained on the C4 dataset.

4 Evaluation and Discussion

We use the publicly available source code of QuEST (Panferov et al., 2025), add the implementation of BBQ, and train on LLaMA (Touvron et al., 2023; Vaswani et al., 2017) models with n non-embedding parameters plus e embedding parameters where $n \in \{30M, 50M, 100M, 200M\}$ and $e \in \{65M, 75M, 100M, 100M\}$. For each model, we pre-train with $100n$ C4 tokens while quantizing the weights and activations of all linear layers to b-bit following QuEST. We compare BBQ against QuEST/LSQ for each $b \in \{1, 2, 3, 4\}$, and present the evaluation quality (perplexity) and weight entropy (bits) in Table 3. We show zero-shot results in Table 9, and training loss curves in Section A.6. Since LSQ does not support 1-bit quantization and LSQ diverges when $n+e = 300M$, we omit such experiments. Each experiment for $n + e \leq 200M$ is conducted on one Nvidia RTX 5090 and lasts for up to 1 day. Each experiment for $n + e = 300M$ is conducted on one Nvidia A100 80GB and lasts for 3.5 days. In total, all experiments took approximately 1.5 GPU months, which is all we can afford in a non-corporate setting with limited access to shared hardware. Lastly, we evaluate BBQ against QuEST, LSQ, ParetoQ-SEQ (Liu et al., 2025), and NormalFloat (Dettmers et al., 2023) on GPT (Brown et al., 2020) models and present our results in Table 4.

Results in Table 3 and 4 suggest BBQ can consistently achieve higher entropy and lower perplexity than QuEST and LSQ at the same precision, and that entropy is a good proxy to prediction quality. A special case is when $b = 1$. In this case, both QuEST and LSQ achieves the maximal entropy of 1 bit. We attribute the performance gain to the fact that Φ from BBQ is smoother than the clip operation from QuEST and LSQ, and therefore have nicer properties for optimization methods like Gradient Descent. In addition, we note that LSQ diverges for LLaMA-200M, which is reflected in its entropy. Notably, when LSQ does not diverge, like in the case of LLaMA-95M and LLaMA-125M, we see that LSQ can achieve an entropy of 3.6 bits for 4-bit precision, 2.74 bits for 3-bit precision, and 1.7 bits for 2-bit precision. However, when LSQ diverges, we see that its entropy drops to 2.73 bits for 4-bit precision, 2.50 bits for 3 bit precision, and 1.63 bits for 2-bit precision. This shows that

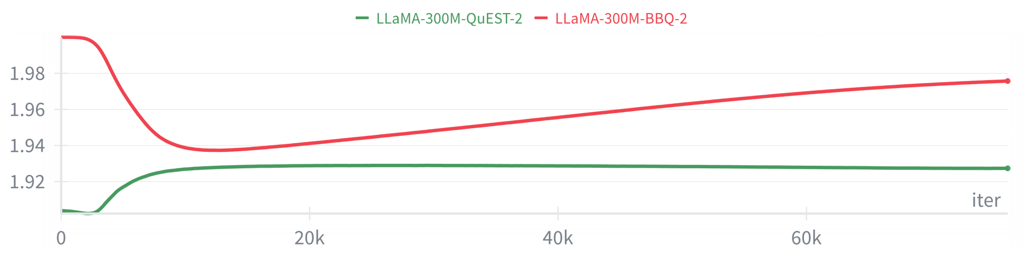

Figure 3: Quantized weight entropy (y-axis) vs. training iterations (x-axis) for LLaMA-300M with 2-bit weight and activations, pre-trained on 20 billion C4 tokens (batched into 80 thousand training iterations). BBQ is red and QuEST is green.

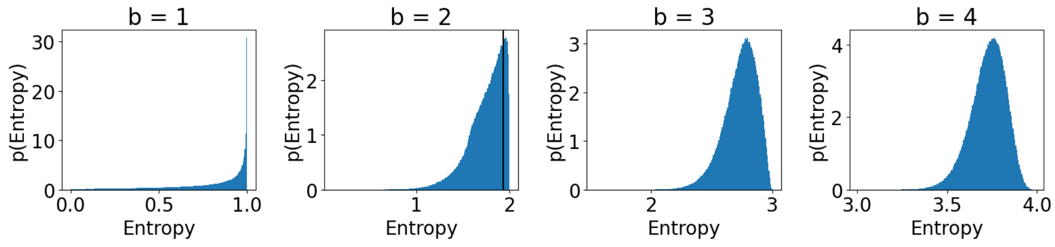

Figure 4: Monte Carlo Estimation of the PDF of the entropy of a 2^b-category discrete distribution. The figures are generated as follows. First, we generate s lists, where each list has 2^b numbers, each i.i.d. sampled from $U(0,1)$. Next, for each of the s lists, we normalize the list such that all 2^b numbers in that list add up to 1. Effectively, this creates s randomly generated discrete distributions. Next, for each list l, we calculate its entropy with the formula $\sum_{i=1}^{2^b} -l[i]\log_2 l[i]$, obtaining a total of s samples of entropy. Lastly, we plot the histogram of entropy to estimate its PDF. We use $s = 1,000,000$ and $b \in \{1,2,3,4\}$. The plots suggest that for $b = 1$, most discrete distributions have entropy > 0.5; for $b = 2$, most discrete distributions have entropy > 1; for $b = 3$, most distributions have entropy > 2; and lastly, for $b = 4$, most distributions have entropy > 3.5.

entropy is a good indicator of a quantized model's prediction quality. Lastly, Table 3 shows BBQ achieves SOTA perplexity for 1-bit quantization and BBQ works for models at multiple scales.

4.1 Entropy and Learning Capacity

In this section, we discuss the implications of Figure 3, which contains the behaviour of weight entropy of BBQ and QuEST during 2-bit training. We note a few observations:

- BBQ can maximize entropy as it can achieve 2 bits of entropy at the beginning of training.
- During training, a BBQ-quantized model can still learn to decrease its entropy if that is (temporarily) the best thing to do.
- Excluding the first 8 thousand iterations when learning rate is warming up, BBQ tends to increase weight entropy for the last 72 thousand iterations. This suggest as more training tokens are presented to the model, more information is accumulated in weights.
- The entropy of QuEST seems to have an empirical ceiling at around 1.93 bits.

By having a lower empirical entropy upper bound of 1.93 bits, QuEST limits the learning capacity of the model, while BBQ does not suffer from such capacity under-utilization. Suppose a model has n parameters, each quantized to b bits. This means the model has a total of 2^{nb} possible unique states. Some of the 2^{nb} states have higher entropy, while others have lower entropy. QAPT's objective is to find a (locally) optimal state out of all 2^{nb} possible states. However, if QuEST enforces its weight

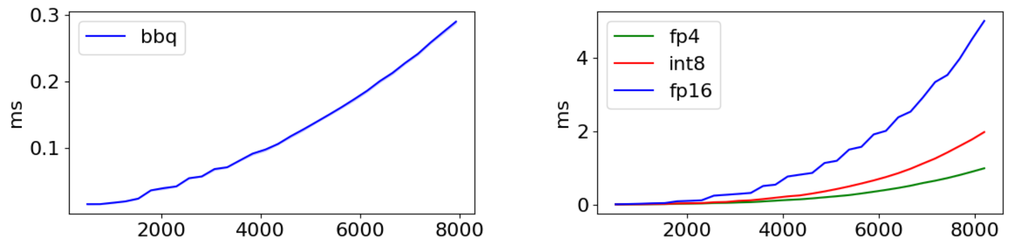

Figure 5: The left figure shows latency of BBQ quantization kernel of an $N \times N$ matrix. The right figure shows the latency of FP4/INT8/FP16 matrix multiplication of two $N \times N$ matrices. The y-axis is latency and the x-axis is N. All latency measurements are conducted on Nvidia RTX 5090.

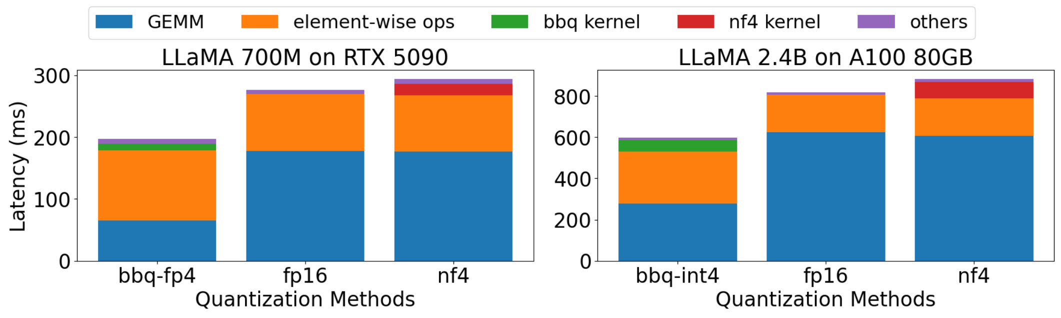

Figure 6: End-to-end latency improvement of BBQ vs. full-precision and NormalFloat 4 on RTX 5090 (a Blackwell GPU that supports fp4 matmul) and A100 (an Ampere GPU that supports int4 matmul) GPUs. For each linear layer, BBQ launches an activation quantization kernel (green region), an fp4/int4 matrix multiplication kernel (part of blue regions), and an element-wise scaling kernel (part of orange region).

entropy to be lower than 1.93 bits, then some of the high-entropy states cannot be explored by the training algorithm. Therefore, the model's learning capacity is effectively reduced.

To numerically quantify the under-utilization of QuEST and the improvement in capacity utilization of BBQ over QuEST, we use Monte Carlo sampling to estimate the PDF of entropy in Figure 4 for $b \in \{1, 2, 3, 4\}$. The figure suggests that regardless of n, most of the 2^{nb} states have medium to high entropy, while very few have low entropy. Since QuEST has an empirical entropy upper bound of 1.93 bits (indicated by the black line in Figure 4 column 2), we estimate from Figure 4 that QuEST's search space is limited to $0.82 \cdot 2^{nb}$ states. On the other hand, BBQ has an entropy upper bound of 2 bits, and therefore has a search space of 2^{nb}, or a 22% gain in search space size over QuEST.

4.2 Inference Speedup

In this section, we discuss our profiling results of the latency of a Triton implementation (shown in Section A.1) of Algorithm 1 on $N \times N$ activation matrices, for $N \in [256, 8192]$. Specifically, $N = 8192$ corresponds to the dimension of LLaMA-65B (Touvron et al., 2023), an important LLM benchmark. We perform latency measurements on Nvidia RTX 5090, a Blackwell GPU with support for MX FP4. Figure 5 left shows that when $N = 8192$, our BBQ quantization kernel takes 0.3 milliseconds to quantize an $N \times N$ activation matrix to FP4. However, Figure 5 right shows the latency saving of FP4 over FP16 is 4 milliseconds when $N = 8192$. Therefore, for models at the scale of LLaMA-65B, BBQ with FP4 quantization leads to 3.7 milliseconds of latency reduction per linear layer. In addition, for the memory-bound token generation phase of inference, using FP4 can still lead to speedup since weights/activations are quantized to FP4 to save memory. In Figure 4.1, we additionally show end-to-end LLaMA inference speedup on both NVIDIA RTX 5090 and NVIDIA A100 80GB, an Ampere GPU. In Figure 4.1, the latency of BBQ quantization kernel is

Hadamard	RMS Norm	PIT	Learnable γ	γ Init	Perplexity	Entropy	Notes
✓	✓	✓	✓	✓	31.34	1.97	BBQ
✗	✓	✓	✓	✓	35.79	1.98	
✓	✗	✓	✓	✓	35.93	1.98	
✓	✓	✗	✗	✗	35.58	1.92	QuEST
✓	✓	✓	✗	✗	138.3	1.92	
✓	✓	✓	✗	✓	31.46	1.98	

Table 5: Ablation of BBQ features, evaluated on LLaMA-95M trained with 3B C4 tokens. The features are Hadamard Transform, RMS Normalization, Probability Integral Transform (PIT or Φ), making γ a learnable parameter, and γ initialization. A ✓ means a feature is present, and ✗ means the feature is not included.

illustrated as a fraction of the overall latency. In general, BBQ shows a 40% speedup over full-precision baseline and a 48% speedup over NF4 quantization.

4.3 Ablation

In this section, we present ablation studies of BBQ's individual features in Table 4.2. Specifically, we study the effects of Hadamard Transform, RMS Normalization, probability integral transform (Φ), having a learnable scale γ, and initializing γ to $\zeta^*\sigma_0$ instead of a dummy value (e.g. 1). Results show Hadamard transform and RMS normalization are two solid features BBQ inherited from QuEST, as removing them from BBQ results in a perplexity increase of 4.45 and 4.59 points, respectively. The probability integral transform (Φ) is meant to replace QuEST's clip function. We see that naively substituting clip with Φ (row 4 compared to row 5) without γ initialization leads to divergence (a perplexity increase from 35.58 to 138.3), but the combination of Φ and γ initialization (row 6) can outperform QuEST (row 4) by a perplexity decrease of 4.12. Lastly, making γ learnable can further reduce the perplexity by a small margin (0.12 perplexity reduction).

5 Limitations

While BBQ can achieve lower perplexity than QuEST and LSQ when a model with limited memory footprint is initialized randomly and trained on a large dataset (**QAPT**), BBQ is, by design, a quantizer that makes no attempt to reduce/bound the Euclidean distance between x and $\hat{x}$, since they live in separate domains. Therefore, when applied to **QAFT**, or finetuning a large model initialized from a full-precision pre-trained checkpoint for a short duration on a small downstream dataset, BBQ's unbounded quantization error will drastically reduce the model quality and cannot catch up to "same-domain" quantizers like QuEST and LSQ within the short time frame of QAFT. For similar reasons, BBQ is not suitable for **PTQ**.

6 Conclusion

In this work, we identify existing SOTA QAPT methods under-utilize learning capacity. While existing ITO quantization methods can maximize entropy, they are not compute-efficient, which limits their applicability to edge devices with energy constraints. Utilizing our key insight that learning is domain-agnostic, we propose BBQ, which performs ITO quantization in the input domain, while returning outputs in a compute-efficient domain. Empirical results shows BBQ outperforms QuEST and LSQ without sacrificing compute efficiency.

7 Reproducibility statement

We will upload our source code as a separate zip file. Our code should allow reviewers to reproduce Table 3 and Figures 3 and 5.

References

Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018. URL `http://arxiv.org/abs/1803.08375`. cite arxiv:1803.08375Comment: 7 pages, 11 figures, 9 tables.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL `https://arxiv.org/abs/1607.06450`.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL `https://arxiv.org/abs/1308.3432`.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning, 2018. URL `https://arxiv.org/abs/1606.04838`.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

Roberto L. Castro, Andrei Panferov, Soroush Tabesh, Oliver Sieberling, Jiale Chen, Mahdi Nikdan, Saleh Ashkboos, and Dan Alistarh. Quartet: Native fp4 training can be optimal for large language models. *CoRR*, abs/2505.14669, May 2025. URL `https://doi.org/10.48550/arXiv.2505.14669`.

Feng Cheng, Cong Guo, Chiyue Wei, Junyao Zhang, Changchun Zhou, Edward Hanson, Jiaqi Zhang, Xiaoxiao Liu, Hai Li, and Yiran Chen. Ecco: Improving memory bandwidth and capacity for llms via entropy-aware cache compression. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ISCA '25, pp. 793–807, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712616. doi: 10.1145/3695053.3731024. URL `https://doi.org/10.1145/3695053.3731024`.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006. ISBN 0471241954.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=shpkpVXzo3h`.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: efficient finetuning of quantized llms. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Jordan Dotzel, Yuzong Chen, Bahaa Kotb, Sushma Prasad, Gang Wu, Sheng Li, Mohamed S. Abdelfattah, and Zhiru Zhang. Learning from students: applying t-distributions to explore accurate and efficient formats for llms. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

DaYou Du, Yijia Zhang, Shijie Cao, Jiaqi Guo, Ting Cao, Xiaowen Chu, and Ningyi Xu. BitDistiller: Unleashing the potential of sub-4-bit LLMs via self-distillation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 102–116, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.7. URL `https://aclanthology.org/2024.acl-long.7/`.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgO66VKDS`.

R. A. Fisher. *Statistical Methods for Research Workers*, pp. 66–70. Springer New York, New York, NY, 1992. ISBN 978-1-4612-4380-9. doi: 10.1007/978-1-4612-4380-9_6. URL `https://doi.org/10.1007/978-1-4612-4380-9_6`.

Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=tcsZt9ZNKD`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL `https://arxiv.org/abs/1606.08415`.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL `https://www.sciencedirect.com/science/article/pii/0893608089900208`.

Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456. JMLR.org, 2015.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: dense-and-sparse quantization. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Tanishq Kumar, Zachary Ankner, Benjamin Frederick Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Re, and Aditi Raghunathan. Scaling laws for precision. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=wg1PCg3CUP`.

Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pp. 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_3. URL `https://doi.org/10.1007/978-3-642-35289-8_3`.

Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. Duquant: Distributing outliers via dual transformation makes stronger quantized LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=mp8u2Pcmqz`.

Jing Liu, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, and Bohan Zhuang. QLLM: Accurate and efficient low-bitwidth quantization for large language models. In *The Twelfth International Conference on Learning Representations*, 2024a. URL `https://openreview.net/forum?id=FIplmUWdm3`.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. LLM-QAT: Data-free quantization aware training for large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 467–484, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.26. URL `https://aclanthology.org/2024.findings-acl.26/`.

Zechun Liu, Changsheng Zhao, Hanxian Huang, Sijia Chen, Jing Zhang, Jiawei Zhao, Scott Roy, Lisa Jin, Yunyang Xiong, Yangyang Shi, Lin Xiao, Yuandong Tian, Bilge Soran, Raghuraman

Krishnamoorthi, Tijmen Blankevoort, and Vikas Chandra. Paretoq: Improving scaling laws in extremely low-bit LLM quantization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=PMSNd8xTHp.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit llms: All large language models are in 1.58 bits, 2024a. URL https://arxiv.org/abs/2402.17764.

Yuexiao Ma, Huixia Li, Xiawu Zheng, Feng Ling, Xuefeng Xiao, Rui Wang, Shilei Wen, Fei Chao, and Rongrong Ji. Affinequant: Affine transformation quantization for large language models. In *ICLR*, 2024b. URL https://openreview.net/forum?id=of2rhALq8l.

Vladimir Malinovskii, Denis Mazur, Ivan Ilin, Denis Kuznedelev, Konstantin Pavlovich Burlachenko, Kai Yi, Dan Alistarh, and Peter Richtárik. PV-tuning: Beyond straight-through estimation for extreme LLM compression. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=YvA8UF0I37.

Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014. ISBN 1461346916.

Nvidia. Nvidia turing gpu architecture, 2018. URL https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

Nvidia. Nvidia ampere ga102 gpu architecture, 2021. URL https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf.

Nvidia. Nvidia rtx blackwell gpu architecture, 2025. URL https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf.

Andrei Panferov, Jiale Chen, Soroush Tabesh, Roberto L. Castro, Mahdi Nikdan, and Dan Alistarh. QuEST: Training accurate LLMs over highly-compressed weights and activation. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*, 2025. URL https://openreview.net/forum?id=ElgAzv9fNk.

Bita Darvish Rouhani, Nitin Garegrat, Tom Savell, Ankit More, Kyung-Nam Han, Ritchie Zhao, Mathew Hall, Eric Chung Jasmine Klar, Yuan Yu, Michael Schulte, Ralph Wittig, Ian Bratt, Nigel Stephens, Jelena Milanovic, John Brothers, Pradeep Dubey, Marius Cornea, Alexander Heinecke, Andres Rodriguez, Martin Langhammer, Summer Deng, Maxim Naumov, Paulius Micikevicius, Michael Siu, and Colin Verrilli. Ocp microscaling formats (mx) specification, 2023. URL https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf.

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. doi: https://doi.org/10.1002/j.1538-7305.1948.tb01338.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=8Wuvhh0LYW.

Xuan Shen, Zhenglun Kong, Changdi Yang, Zhaoyang Han, Lei Lu, Peiyan Dong, Cheng Lyu, Chih hsiang Li, Xuehang Guo, Zhihao Shu, Wei Niu, Miriam Leeser, Pu Zhao, and Yanzhi Wang. Edgeqat: Entropy and distribution guided quantization-aware training for the acceleration of lightweight llms on the edge. *CoRR*, abs/2402.10787, 2024. URL https://doi.org/10.48550/arXiv.2402.10787.

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL `https://doi.org/10.1145/3315508.3329973`.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers amp; distillation through attention. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10347–10357. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/touvron21a.html`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL `https://arxiv.org/abs/2302.13971`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Haocheng Xi, ChangHao Li, Jianfei Chen, and Jun Zhu. Training transformers with 4-bit integers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=H9hWlfMT6O`.

Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen-Kuang Chen, and Fengbo Ren. Learning in the frequency domain. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1737–1746, 2020. doi: 10.1109/CVPR42600.2020.00181.

Kohei Yamamoto. Learnable Companding Quantization for Accurate Low-bit Neural Networks . In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5027–5036, Los Alamitos, CA, USA, June 2021. IEEE Computer Society. doi: 10.1109/CVPR46437.2021.00499. URL `https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.00499`.

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, and Kurt Keutzer. Hawq-v3: Dyadic neural network quantization. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11875–11886. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/yao21a.html`.

Davis Yoshida. Nf4 isn't information theoretically optimal (and that's good), 2023. URL `https://arxiv.org/abs/2306.06965`.

A Appendix

A.1 Kernel Implementation

In this section, we present an example implementation of Algorithm 1 using the Triton (Tillet et al., 2019) programming language.

Nvidia GPUs provide extensive support for low-precision tensor core operations, allowing expensive high-precision matrix multiplications to be replaced with low-precision alternatives, leading to higher compute throughput and lower energy consumption. For example, the Nvidia Blackwell architecture (Nvidia, 2025) provides tensor core support for the MX FP4(Rouhani et al., 2023) data type. As another example, the Turing (Nvidia, 2018) and Ampere (Nvidia, 2021) architectures provide support for the INT4 data type. We list in Table 6 the representable values for both data types.

Binary Representation	Signed INT4	MX FP4
0b0000	0	0
0b0001	1	0.5
0b0010	2	1
0b0011	3	1.5
0b0100	4	2
0b0101	5	3
0b0110	6	4
0b0111	7	6
0b1000	-8	0
0b1001	-7	-0.5
0b1010	-6	-1
0b1011	-5	-1.5
0b1100	-4	-2
0b1101	-3	-3
0b1110	-2	-4
0b1111	-1	-6

Table 6: Signed INT4 vs. MXFP4

Table 7 shows the values that BBQ can represent.

Precision	Possible Values of q
4	-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7
3	-4,-3,-2,-1,0,1,2,3
2	-1.5,-0.5,0.5,1.5
1	-0.5,0.5

Table 7: Values that BBQ can represent. As shown, for $b = 4$, BBQ can be encoded as INT4. For $b = 3$, BBQ can be encoded as INT4 or FP4. For $b \in \{1, 2\}$, BBQ can be encoded as FP4.

Since $b = 3$ can be encoded by both INT4 and FP4, we present an example triton kernel for BBQ with 3-bit quantization following Algorithm 1.

```
@triton.jit
def bbq(
    x_ptr,
    h_ptr,
    rsigma_ptr,
    qmz_ptr,
    ROW_SIZE: tl.constexpr,
    BLOCKSIZE: tl.constexpr=128,
    DTYPE: tl.constexpr = "int4"
):
    rid = tl.program_id(axis=0)
    cid = tl.program_id(axis=1)
    xrids = rid * BLOCKSIZE + tl.arange(0, BLOCKSIZE)[:, None]
    xcids = cid * BLOCKSIZE + tl.arange(0, BLOCKSIZE)[None, :]
    xids = xrids * ROW_SIZE + xcids

    hrids = tl.arange(0, BLOCKSIZE)[:, None]
    hcids = tl.arange(0, BLOCKSIZE)[None, :]
    hids = hrids * BLOCKSIZE + hcids

    # load a block of x
    x = tl.load(x_ptr + xids)
    # load pre-computed hadamard matrix shared across the entire kernel
    h = tl.load(h_ptr + hids)
    # load reciprocal of sigma
    rsigma = tl.load(rsigma_ptr)

    # hadamard transform
    v = tl.dot(x, h) * rsigma

    # accelerated normal CDF and rounding
    qmz = normal_cdf_and_round_3bit(v, DTYPE)

    # pack two 4-bit data type into a single byte
    shift = tl.arange(0, 2)[None, None, :] * 4
    qmz = tl.reshape(qmz, (BLOCKSIZE, BLOCKSIZE // 2, 2)) << shift
    qmz = tl.xor_sum(qmz, axis=-1, keep_dims=False)

    # write quantized data to memory
    qmzrids = rid * BLOCKSIZE + tl.arange(0, BLOCKSIZE)[:, None]
    qmzcids = cid * (BLOCKSIZE // 2) + tl.arange(0, BLOCKSIZE // 2)[None, :]
    qmzids = qmzrids * (ROW_SIZE // 2) + qmzcids
    tl.store(qmz_ptr + qmzids, qmz)
```

Next, we show the definition of function normal_cdf_and_round_3bit. We pre-compute $\Phi^{-1}\left(\frac{i}{2^b}\right)$ for all possible values of i, and use a binary search method to decide which i should be the result of quantization, and lastly, directly use the binary representation of $i - 2^{b-1} - z$ according to Table 6. For 3-bit quantization, the 8 pre-computed values of $\Phi^{-1}\left(\frac{i}{2^b}\right)$, for $i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$, are {-∞, -1.1503493803760083, -0.6744897501960818, -0.3186393639643752, 0.0, 0.3186393639643752, 0.6744897501960818, 1.1503493803760083}. The corresponding values of $i - 2^{b-1} - z$ are $\{-4, -3, -2, -1, 0, 1, 2, 3\}$.

```
@triton.jit
def normal_cdf_and_round_3bit(v: tl.tensor, DTYPE: tl.constexpr):
    if DTYPE == "int4":
        qmz = tl.where(
            v >= 0.0,
            tl.where(
                v >= 0.6744897501960818,
                tl.where(v >= 1.1503493803760083, 0b0011, 0b0010),
                tl.where(v >= 0.3186393639643752, 0b0001, 0b0000),
            ),
            tl.where(
                v >= -0.6744897501960818,
                tl.where(v >= -0.3186393639643752, 0b1111, 0b1110),
                tl.where(v >= -1.1503493803760083, 0b1101, 0b1100),
            )
        ).to(tl.int8)
```

```
            return qmz
        elif DTYPE == "fp4":
            qmz = tl.where(
                v >= 0.0,
                tl.where(
                    v >= 0.6744897501960818,
                    tl.where(v >= 1.1503493803760083, 0b0101, 0b0100),
                    tl.where(v >= 0.3186393639643752, 0b0010, 0b0000),
                ),
                tl.where(
                    v >= -0.6744897501960818,
                    tl.where(v >= -0.3186393639643752, 0b1010, 0b1100),
                    tl.where(v >= -1.1503493803760083, 0b1101, 0b1110),
                )
            ).to(tl.int8)
            return qmz
```

A.2 Ablation on Exponential Moving Average of σ

Results in Table 3 are conducted without keeping an EMA of $1/\sigma$ during training and use it for inference. In this section, we show both variants of BBQ achieve identical perplexity.

	4-bit	3-bit	2-bit	1-bit
BBQ	25.51	26.55	31.35	49.8
BBQ with EMA of Reciprocal of σ	25.4	26.55	31.43	49.72

Table 8: Perplexity of BBQ vs BBQ with an exponential moving average of $1/\sigma$, on LLaMA-95M pre-trained with 3 billion C4 tokens.

A.3 Zero-shot Results

In this section, we present the zero-shot evaluation perplexity in Table 9, which corresponds to the models pre-trained in Table 3.

Model	Dataset	Params n+e	Method	Bits WA	Perplexity
LLaMA	wikitext	95M	None	16	56.11
LLaMA	wikitext	95M	BBQ	4	**57.87**
LLaMA	wikitext	95M	QuEST	4	58.91
LLaMA	wikitext	95M	LSQ	4	61.62
LLaMA	wikitext	95M	BBQ	3	**60.18**
LLaMA	wikitext	95M	QuEST	3	64.21
LLaMA	wikitext	95M	LSQ	3	68.35
LLaMA	wikitext	95M	BBQ	2	**70.19**
LLaMA	wikitext	95M	QuEST	2	77.32
LLaMA	wikitext	95M	LSQ	2	80.11
LLaMA	wikitext	95M	BBQ	1	**109.66**
LLaMA	wikitext	95M	QuEST	1	172.77
LLaMA	wikitext	125M	None	16	50.26
LLaMA	wikitext	125M	BBQ	4	**50.50**
LLaMA	wikitext	125M	QuEST	4	51.31
LLaMA	wikitext	125M	LSQ	4	53.72
LLaMA	wikitext	125M	BBQ	3	**50.94**
LLaMA	wikitext	125M	QuEST	3	54.37
LLaMA	wikitext	125M	LSQ	3	58.61
LLaMA	wikitext	125M	BBQ	2	**60.47**
LLaMA	wikitext	125M	QuEST	2	69.45
LLaMA	wikitext	125M	LSQ	2	70.61
LLaMA	wikitext	125M	BBQ	1	97.34
LLaMA	wikitext	125M	QuEST	1	**180.33**
LLaMA	wikitext	200M	None	16	40.42
LLaMA	wikitext	200M	BBQ	4	42.56
LLaMA	wikitext	200M	QuEST	4	**41.67**
LLaMA	wikitext	200M	LSQ	4	4133
LLaMA	wikitext	200M	BBQ	3	**41.82**
LLaMA	wikitext	200M	QuEST	3	43.95
LLaMA	wikitext	200M	LSQ	3	426.8
LLaMA	wikitext	200M	BBQ	2	**49.53**
LLaMA	wikitext	200M	QuEST	2	54.12
LLaMA	wikitext	200M	LSQ	2	208.5
LLaMA	wikitext	200M	BBQ	1	**80.34**
LLaMA	wikitext	200M	QuEST	1	123.19
LLaMA	wikitext	300M	None	16	34.95
LLaMA	wikitext	300M	BBQ	4	38.57
LLaMA	wikitext	300M	QuEST	4	**35.26**
LLaMA	wikitext	300M	BBQ	3	**36.77**
LLaMA	wikitext	300M	QuEST	3	37.26
LLaMA	wikitext	300M	BBQ	2	**41.24**
LLaMA	wikitext	300M	QuEST	2	45.28

Table 9: Zero-shot wikitext perplexity.

A.4 BBQ vs. QuEST

In this section, we illustrate the difference between BBQ and QuEST in Figure 7.

- BBQ and QuEST share the Hadamard transform (①) and RMS normalization (②) to reshape the distribution to better match the standard Gaussian.
- In step ③, QuEST scales by α^*, shifts by 0.5, and uses the clip function to limit values within a finite range. BBQ uses the Φ function to limit values within a finite range, while simultaneously maximizing the entropy.

- In step ④, both methods use uniform quantization to quantize data. Both methods ensure the quantized data can be stored in a compute-efficient data type (INT4 or FP4).
- In step ⑤, BBQ applies linear scaling, while QuEST reverses all of the operations it did before (RMS Normalization, scaling by α^*, and shifting by 0.5), as QuEST is a "same-domain" quantizer. BBQ is a "cross-domain" quantizer, so BBQ does not reverse its previous operation such as Φ, shifting by $2^b - 1$ and z, multiplication by 2^b, division by σ etc. Instead, BBQ applies linear scaling to scale its range to be within $[-\gamma, \gamma]$ where γ is a learnable parameter.
- In step ⑥, QuEST reverses its Hadamard transform (a unitary operation). BBQ does not reverse the Hadamard transform because it is a "cross-domain" quantizer.

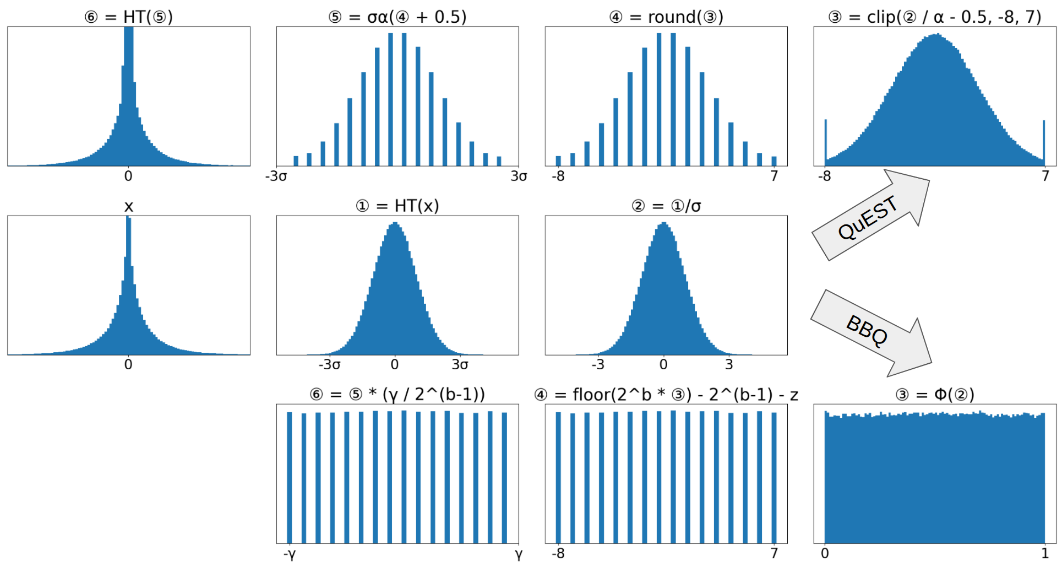

Figure 7: BBQ vs. QuEST.

A.5 Extension to Vision Models

While BBQ is designed for language models, we discuss and evaluate a potential extension of BBQ to vision models. When we naively apply BBQ to vision models without any modification, we notice, as shown in Figure 8 right, a subset of learned γ have magnitudes extremely close to 0. Since every γ is assigned to a weight channel, if γ is small, weights from the corresponding channel suffers from gradient vanishing. As shown in Figure 8 left, language models do not suffer from such γ collapsing. Instead of making γ a learnable parameter and initializing γ to $\zeta\sigma_0$ as discussed in Section 3.5, we propose a BBQ variant, BBQ-Vision, that instead dynamically calculates the value of γ as $\zeta\sigma$ on every forward pass. BBQ-Vision prevents γ from collapsing to 0, since for γ to be 0, all weights in that channel must also be 0. In addition, BBQ-Vision still preserves some degree of learnability as the network can modify weights/activations to indirectly control the value of σ. We compare BBQ-Vision against QuEST and LSQ on DeiT (Touvron et al., 2021) and Resnet (He et al., 2015) with various sizes and show our results in Table 10. Our preliminary results show BBQ-Vision can outperform QuEST and LSQ.

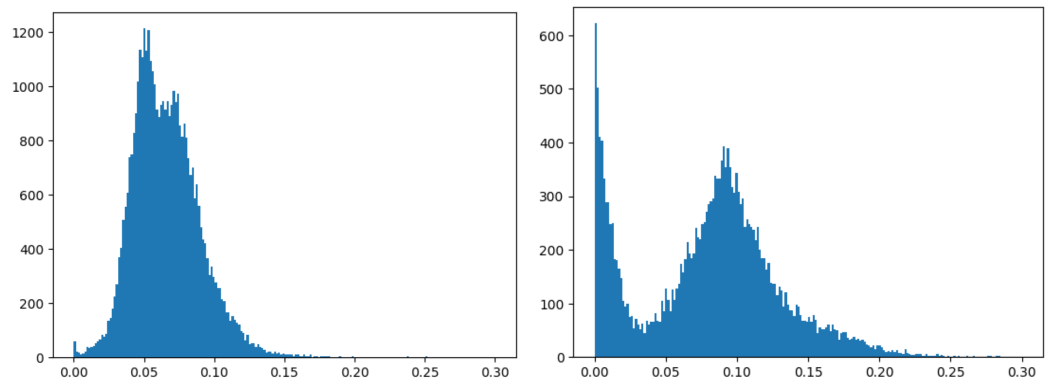

Figure 8: Histogram of learned γ on the 2-bit language model LLaMA-95M (left) and 2-bit vision model DeIT-Tiny (right). The x-axis is the value of γ and the y-axis is its frequency.

Model	Params	Dataset	Bits WA	BBQ-Vision	QuEST	LSQ
DEIT-T	5M	Imagenet-100	4	**3.85/2.84**/74.68	3.61/2.91/74.3	3.67/2.88/**74.88**
DEIT-T	5M	Imagenet-100	3	**2.93/2.89/74.24**	2.78/2.98/72.2	2.81/3.01/71.36
DEIT-T	5M	Imagenet-100	2	**1.94/3.05/70.16**	1.93/3.18/67.24	1.90/3.23/65.32
DEIT-T	5M	Imagenet-100	1	**1.00/3.52/53.54**	**1.00**/3.67/47.55	-/-/-
DEIT-S	20M	Imagenet-100	4	**3.82**/2.53/79.46	3.61/**2.51**/80.00	3.57/2.52/**80.04**
DEIT-S	20M	Imagenet-100	3	**2.91/2.48/80.88**	2.78/2.62/79.34	2.74/2.64/78.96
DEIT-S	20M	Imagenet-100	2	**1.93/2.76/77.02**	1.93/2.82/76.18	1.83/2.87/74.42
DEIT-S	20M	Imagenet-100	1	**1.00/3.18/66.16**	1.00/3.37/59.94	-/-/-
Resnet-10	5M	Imagenet-100	1	**1.00/3.37/63.6**	1.00/3.40/62.4	-/-/-
Resnet-18	11M	Imagenet-100	1	**1.00/3.12/73.13**	1.00/3.14/72.28	-/-/-

Table 10: Entropy/Training Cross Entropy Loss/Evaluation Accuracy of vision models pretrained on vision datasets

A.6 Validation Loss vs. Training Progress

In this section we present validation loss vs. training iteration curves for all of our experiments in Section 4. For visualization purposes, we only show the validation loss for the last 90% iterations, hide the first 10% iterations when learning rate is warming up, and don't show loss curves for methods that diverged. For all figures, the y-axis is the validation cross entropy loss, and x-axis is iterations (batches). For all experiments, we quantize both weights and activations of linear layers to b bits. In addition, for all QuEST experiments, we use a trust factor of $T = \alpha^*/(2^b - 1)$ as discuss in QuEST (Panferov et al., 2025).

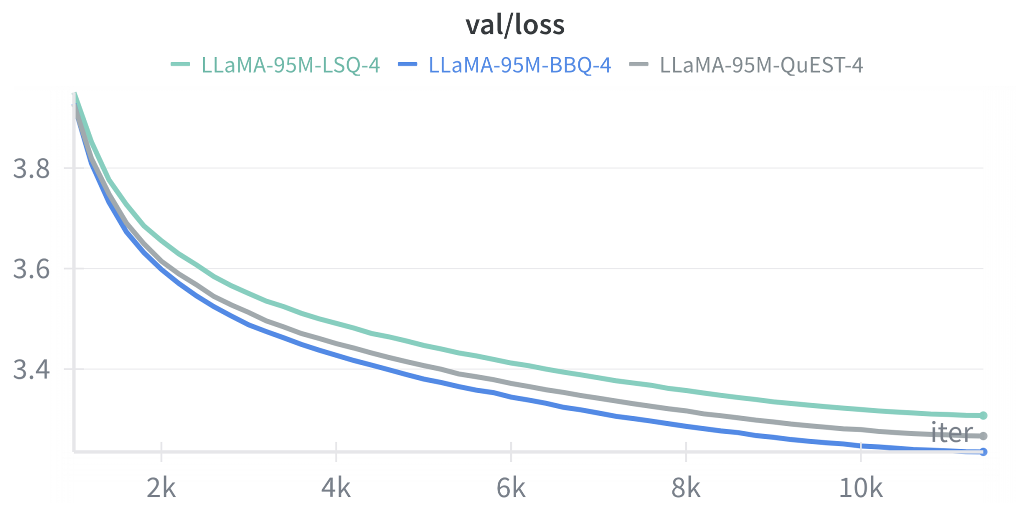

Figure 9: LLaMA-95M (4-bit) pre-trained on 3 billion C4 tokens (batched over 12 thousand iterations). LSQ is green, QuEST is gray, and BBQ is blue.

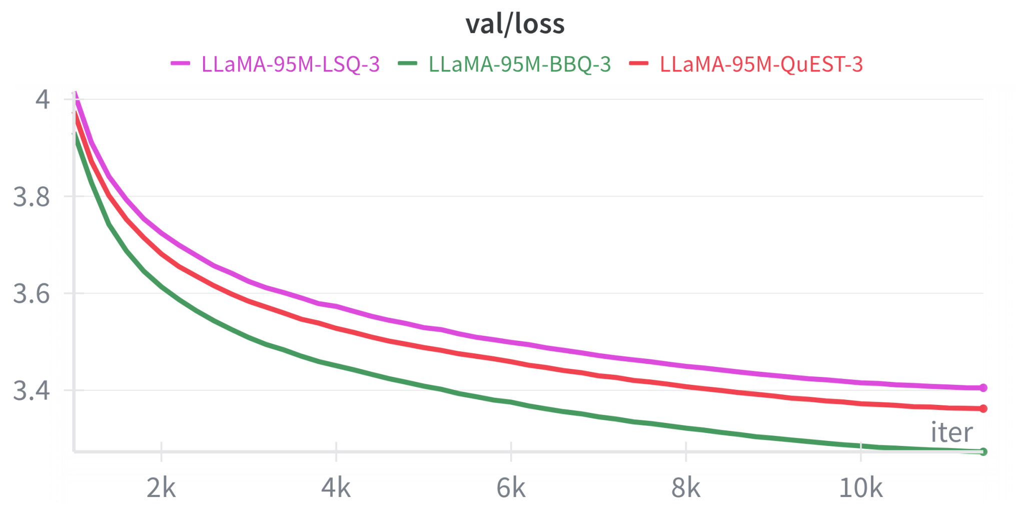

Figure 10: LLaMA-95M (3-bit) pre-trained on 3 billion C4 tokens (batched over 12 thousand iterations). LSQ is pink, QuEST is red, and BBQ is green.

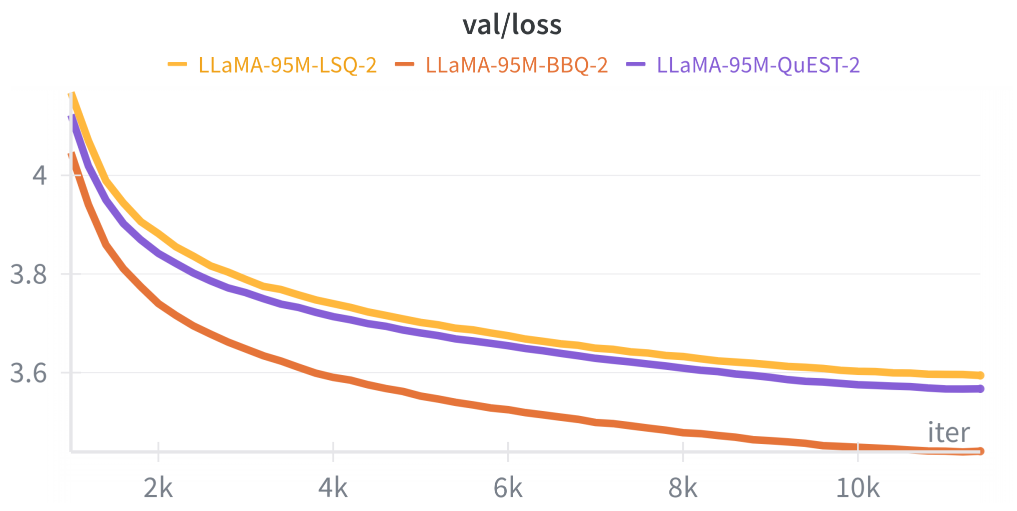

Figure 11: LLaMA-95M (2-bit) pre-trained on 3 billion C4 tokens (batched over 12 thousand iterations). LSQ is yellow, QuEST is purple, and BBQ is orange.

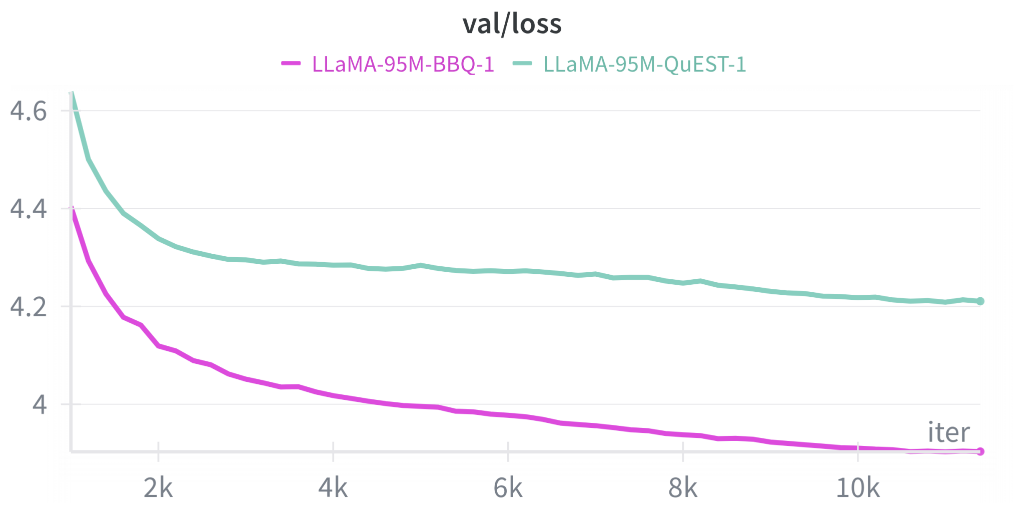

Figure 12: LLaMA-95M (1-bit) pre-trained on 3 billion C4 tokens (batched over 12 thousand iterations). QuEST is purple and BBQ is orange.

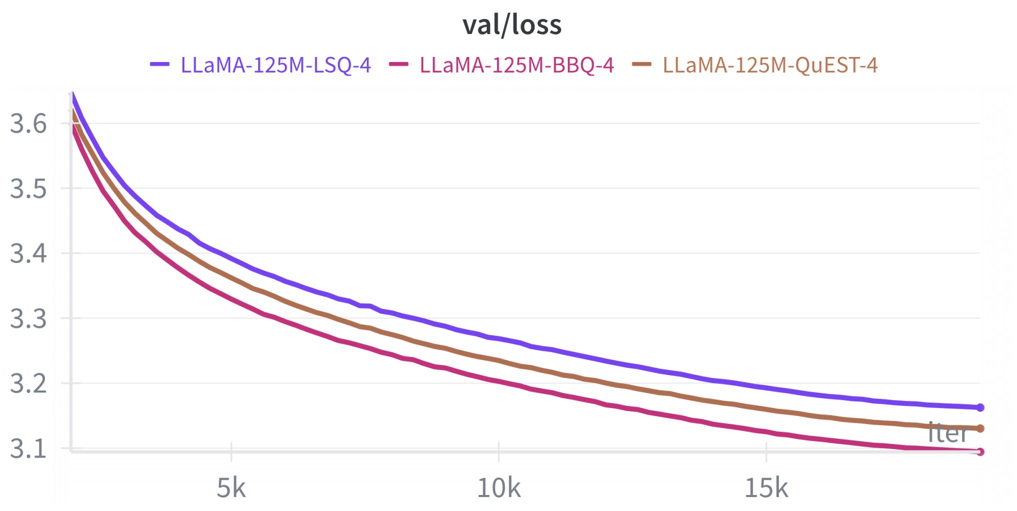

Figure 13: LLaMA-125M (4-bit) pre-trained on 5 billion C4 tokens (batched over 20 thousand iterations). LSQ is red, QuEST is brown, and BBQ is pink.

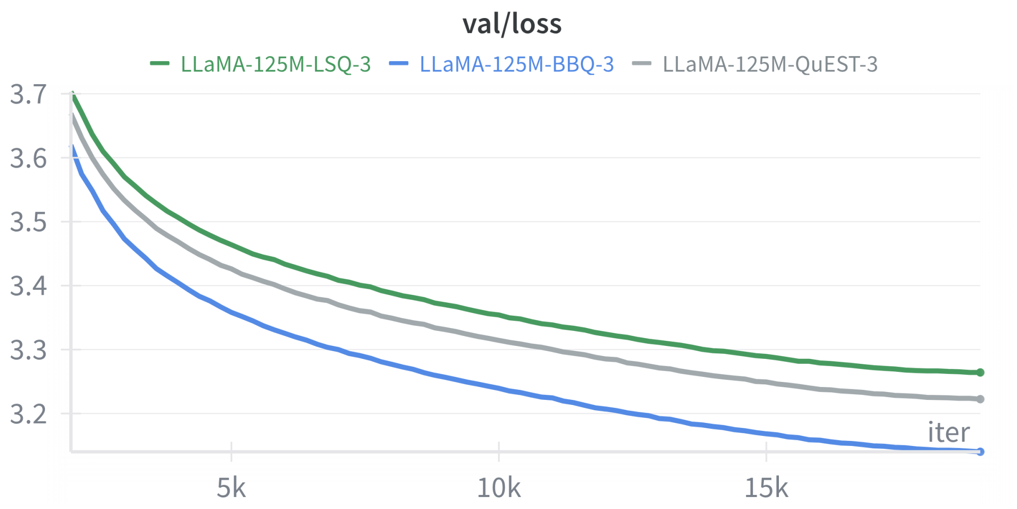

Figure 14: LLaMA-125M (3-bit) pre-trained on 5 billion C4 tokens (batched over 20 thousand iterations). LSQ is green, QuEST is gray, and BBQ is blue.

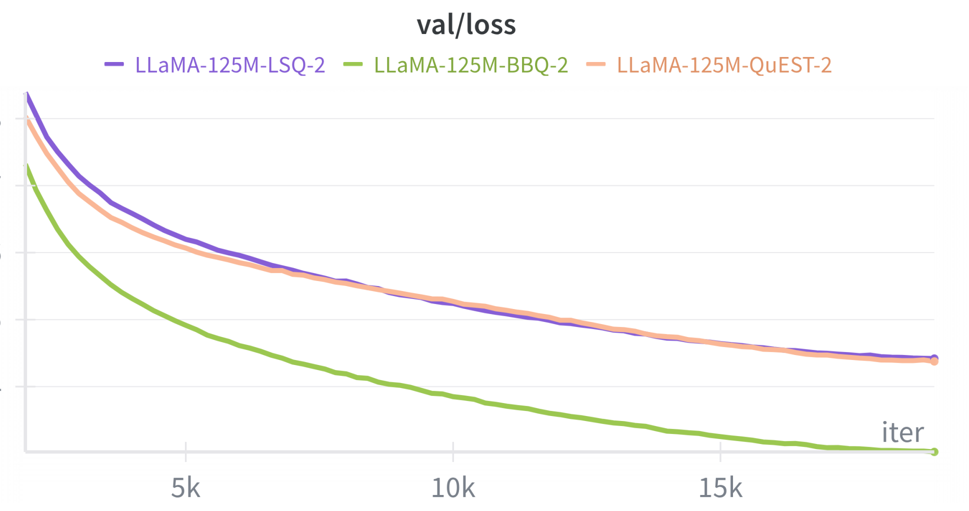

Figure 15: LLaMA-125M (2-bit) pre-trained on 5 billion C4 tokens (batched over 20 thousand iterations). LSQ is purple, QuEST is orange, and BBQ is green.

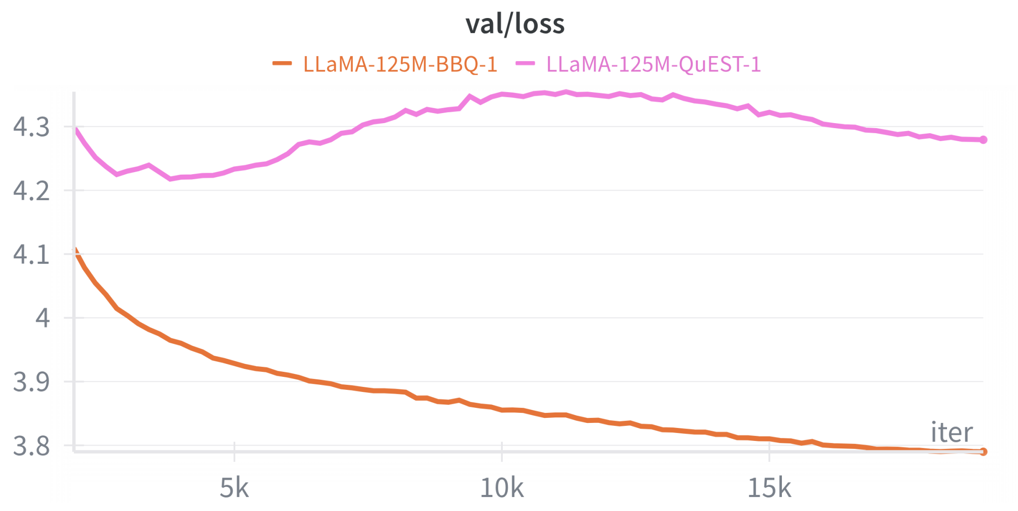

Figure 16: LLaMA-125M (1-bit) pre-trained on 5 billion C4 tokens (batched over 20 thousand iterations). QuEST is pink and BBQ is orange.

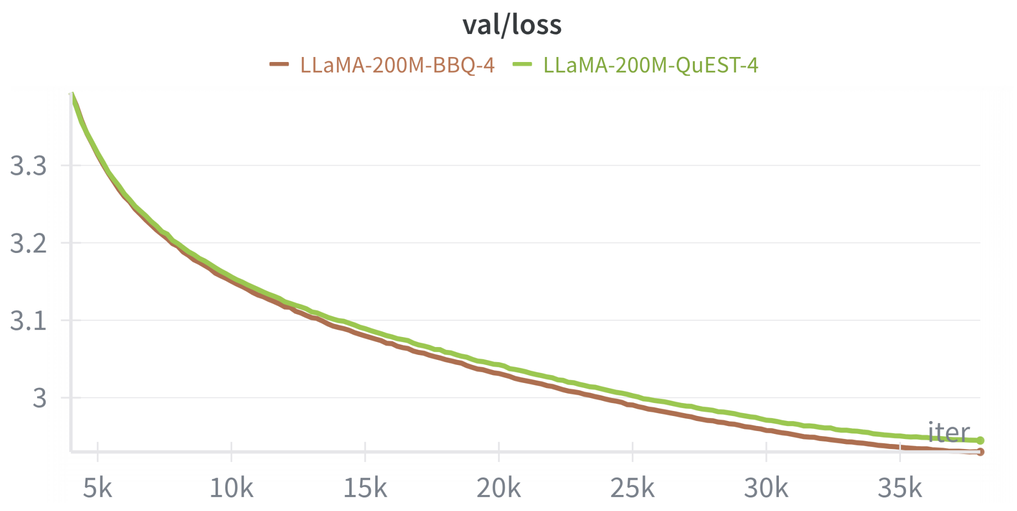

Figure 17: LLaMA-200M (4-bit) pre-trained on 10 billion C4 tokens (batched over 40 thousand iterations). QuEST is green and BBQ is brown.

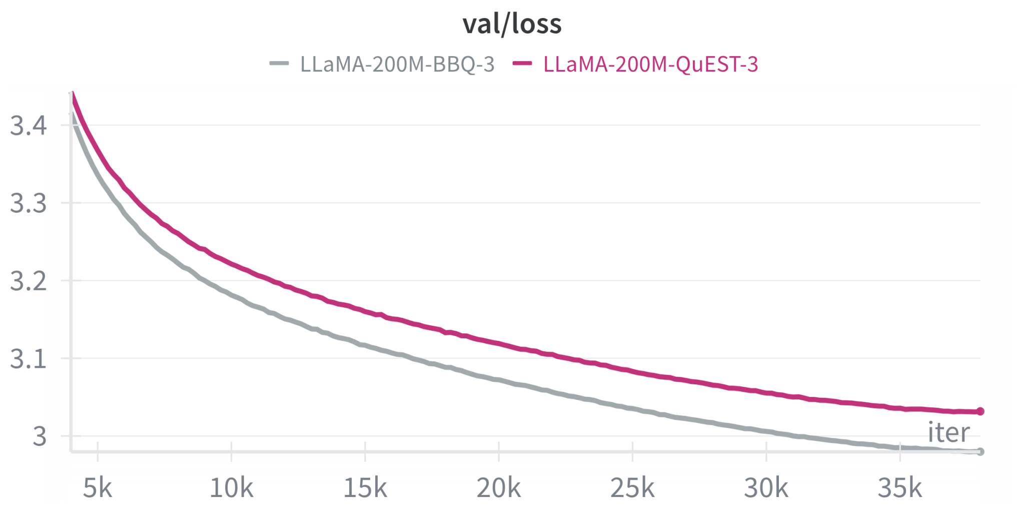

Figure 18: LLaMA-200M (3-bit) pre-trained on 10 billion C4 tokens (batched over 40 thousand iterations). QuEST is pink and BBQ is gray.

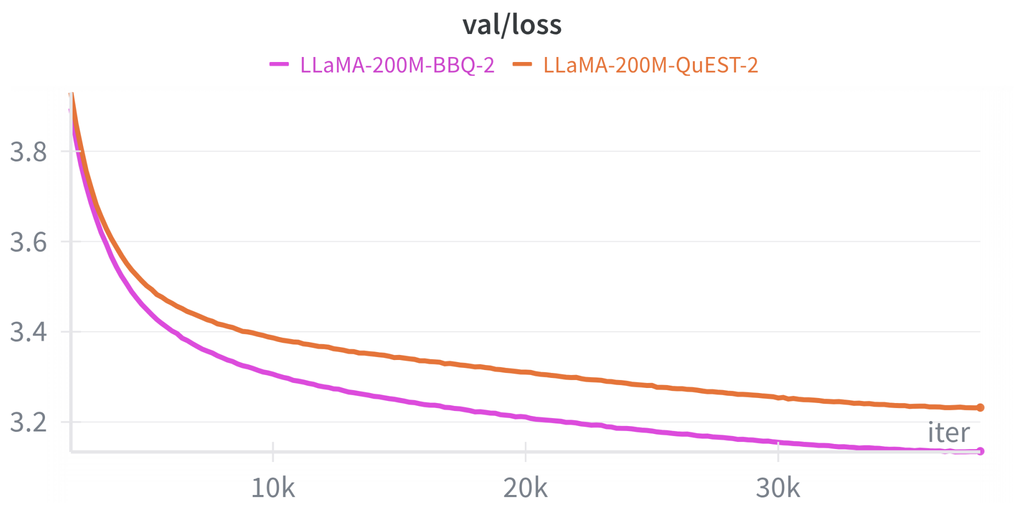

Figure 19: LLaMA-200M (2-bit) pre-trained on 10 billion C4 tokens (batched over 40 thousand iterations). QuEST is orange and BBQ is pink.

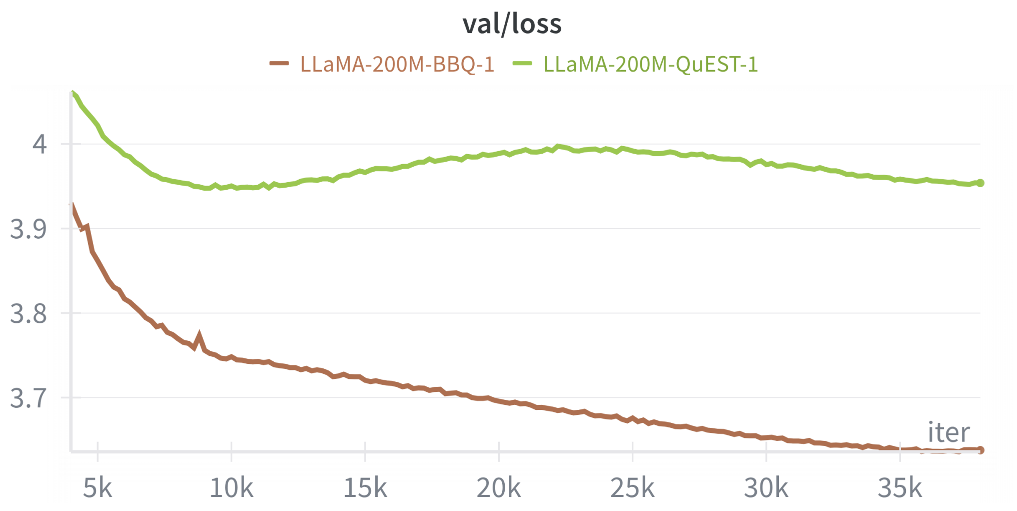

Figure 20: LLaMA-200M (1-bit) pre-trained on 10 billion C4 tokens (batched over 40 thousand iterations). QuEST is green and BBQ is brown.

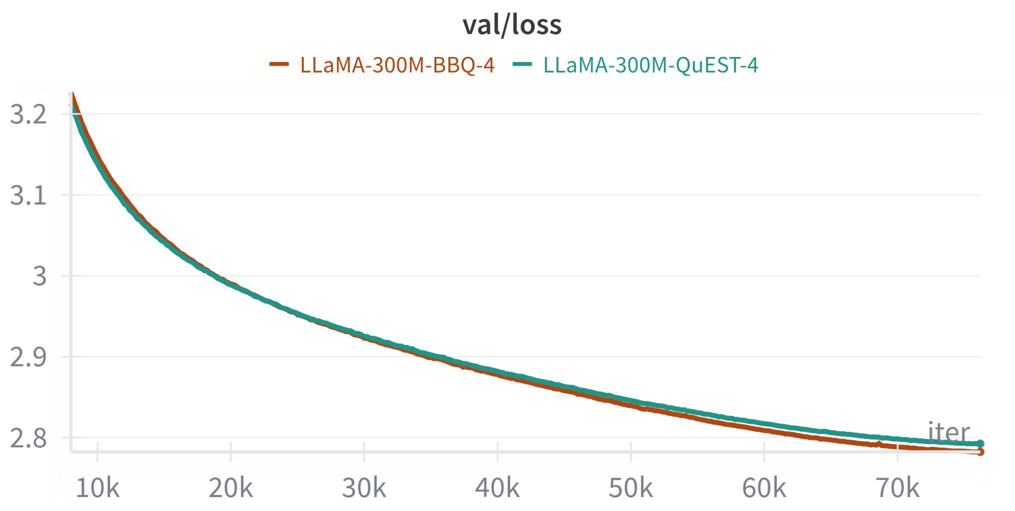

Figure 21: LLaMA-300M (4-bit) pre-trained on 20 billion C4 tokens (batched over 80 thousand iterations). QuEST is green and BBQ is brown.

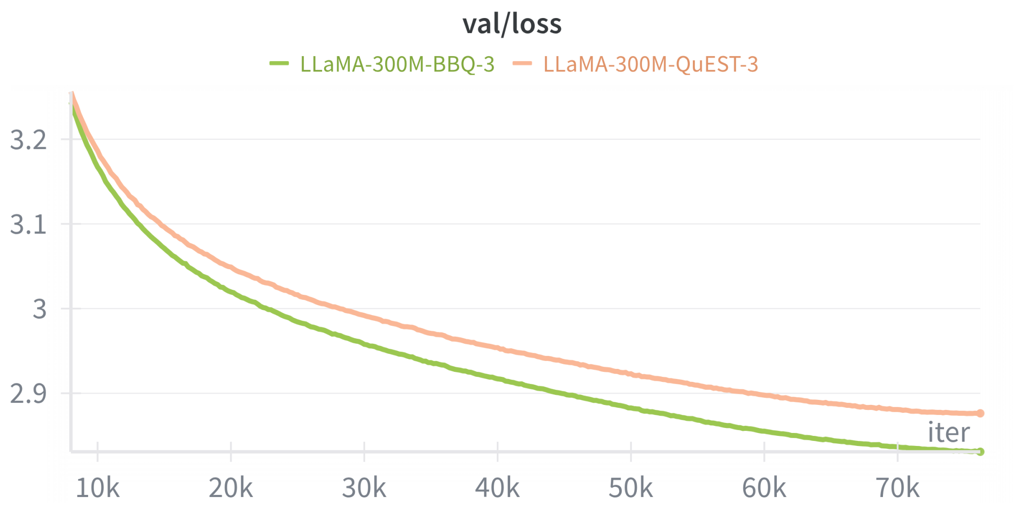

Figure 22: LLaMA-300M (3-bit) pre-trained on 20 billion C4 tokens (batched over 80 thousand iterations). QuEST is orange and BBQ is green.

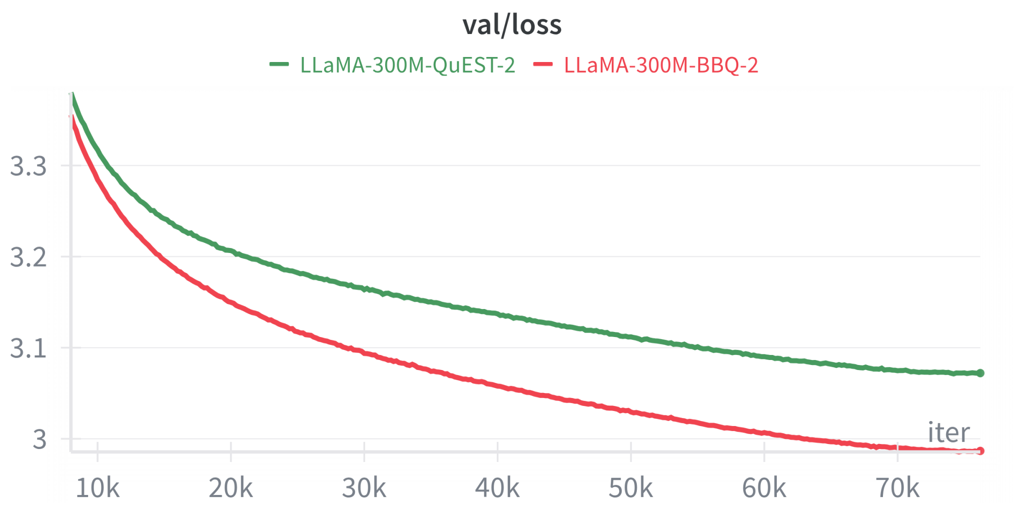

Figure 23: LLaMA-300M (2-bit) pre-trained on 20 billion C4 tokens (batched over 80 thousand iterations). QuEST is green and BBQ is red.