Journal of Machine Learning Research 26 (2025) 1-29 Submitted 11/24; Published 6/25

BitNet: 1-bit Pre-training for Large Language Models

Hongyu Wang[*] WANGHONGYU22@MAILS.UCAS.AC.CN
Shuming Ma[*] SHUMMA@MICROSOFT.COM
Lingxiao Ma, Lei Wang, Wenhui Wang, Li Dong, Shaohan Huang
Huaijie Wang, Jilong Xue, Ruiping Wang, Yi Wu, Furu Wei

Editor: Kai-Wei Chang

Abstract

The increasing size of large language models (LLMs) has posed challenges for deployment and raised concerns about environmental impact due to high energy consumption. Previous research typically applies quantization after pre-training. While these methods avoid the need for model retraining, they often cause notable accuracy loss at extremely low bit-widths. In this work, we explore the feasibility and scalability of 1-bit pre-training. We introduce BitNet b1 and BitNet b1.58, the scalable and stable 1-bit Transformer architecture designed for LLMs. Specifically, we introduce BitLinear as a drop-in replacement of the nn.Linear layer in order to train 1-bit weights from scratch. Experimental results show that BitNet b1 achieves competitive performance, compared to state-of-the-art 8-bit quantization methods and FP16 Transformer baselines. With the ternary weight, BitNet b1.58 matches the half-precision Transformer LLM with the same model size and training tokens in terms of both perplexity and end-task performance, while being significantly more cost-effective in terms of latency, memory, throughput, and energy consumption. More profoundly, BitNet defines a new scaling law and recipe for training new generations of LLMs that are both high-performance and cost-effective. It enables a new computation paradigm and opens the door for designing specific hardware optimized for 1-bit LLMs.

Keywords: Natural Language Processing, Large Language Models, 1-bit Pre-training, Efficiency, Model Architecture

1. Introduction

The rapid growth of large language models (Brown et al., 2020; OpenAI, 2023; Chowdhery et al., 2022; Anil et al., 2023; Touvron et al., 2023a,b) has led to significant improvements in various tasks. However, it is expensive to host large language models due to the high inference costs and energy consumption. As the size of these models grows, the memory bandwidth required for accessing and processing the model parameters becomes a major bottleneck, limiting the overall inference performance. Moreover, when deploying these models on distributed systems or multi-device platforms, the inter-device communication overhead can significantly impact the inference latency and energy consumption. Model quantization has

[*]. Equal contribution. Hongyu Wang and Ruiping Wang are with the Key Laboratory of AI Safety of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS and University of Chinese Academy of Sciences, Beijing. Shuming Ma, Lingxiao Ma, Wenhui Wang, Li Dong, Shaohan Huang, Jilong Xue and Furu Wei are with Microsoft Research. Lei Wang is with University of Chinese Academy of Sciences. Huaijie Wang and Yi Wu are with the Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing. Corresponding author: Hongyu Wang.

emerged as a promising solution, as it can significantly reduce the memory footprint and computational cost of large-scale models while maintaining competitive performance.

Most existing quantization approaches for large language models are post-training (Dettmers et al., 2022; Frantar et al., 2023; Chee et al., 2023; Xiao et al., 2023; Shang et al., 2023; Liu et al., 2023b). They are simple and easy to apply since it does not require any changes to the training pipeline or retraining the model. However, it will result in a more significant loss of accuracy especially when the precision goes lower, because the model is not optimized for the quantized representation during training (Chee et al., 2023; Tseng et al., 2024). Furthermore, although post-training quantization methods can maintain performance comparable to half-precision baselines on language modeling tasks, they often lead to significant degradation on more challenging tasks, such as mathematical reasoning (Li et al., 2025). Another strand of quantizing deep neural networks is quantization-aware training (Peng et al., 2023). Compared to post-training, it typically results in better accuracy, as the model is trained to account for the reduced precision from the beginning. Moreover, it allows the model to continue-train or do fine-tuning, which is essential for large language models. The challenge of quantization-aware training mainly lies in optimization, i.e., the model becomes more difficult to converge as the precision goes lower. Besides, it is unknown whether quantization-aware training follows the scaling law of neural language models.

In this work, we focus on the ultra low-bit (i.e., 1-bit) quantization, applied to large language models. Previous studies on binarized or ternarized neural networks (Rastegari et al., 2016; Bulat and Tzimiropoulos, 2019; Liu et al., 2021; Zhang et al., 2022; Liu et al., 2023a) have mostly revolved around convolutional neural networks. Recently, there has been some research on 1-bit Transformers. However, these studies have focused on machine translation (Zhang et al., 2023) or BERT pretraining (Liu et al., 2022; Bai et al.; Qin et al., 2022; Zhang et al., 2020), which is quite different from large language models. For example, machine translation employs an encoder-decoder architecture, BERT pretraining utilizes a bidirectional encoder, and large language models use a unidirectional decoder. Furthermore, large language models are typically scaled up to a much larger model size, while BERT and machine translation models do not undergo such extensive scaling.

To the best of our knowledge, this work is the first to investigate 1-bit pre-training for large language models. We propose BitNet b1 and BitNet b1.58, the 1-bit and 1.58-bit Transformer architecture for large language models, which aims to scale efficiently in terms of both memory and computation. BitNet employs extremely low-precision weights and quantized activations, while maintaining high precision for the optimizer states and gradients during training. Our approach is designed to be scalable and stable, with the ability to handle large language models efficiently. The implementation of the BitNet architecture is quite simple, requiring only the replacement of linear projections (i.e., *nn.Linear* in PyTorch) in the Transformer. Furthermore, it complements other acceleration methods for large language models, such as PagedAttention (Kwon et al., 2023), FlashAttention (Dao et al., 2022; Dao, 2023), and speculative decoding (Leviathan et al., 2023).

We evaluate BitNet b1 and BitNet b1.58 on a range of language modeling benchmarks, comparing with state-of-the-art quantization methods and FP16 Transformers. Experimental results demonstrate that BitNet b1 achieves competitive performance. Furthermore, our experiments show that BitNet b1.58 can match half-precision (i.e., FP16 or BF16) baselines in terms of both perplexity and end-task performance, starting from a 3B size, when using

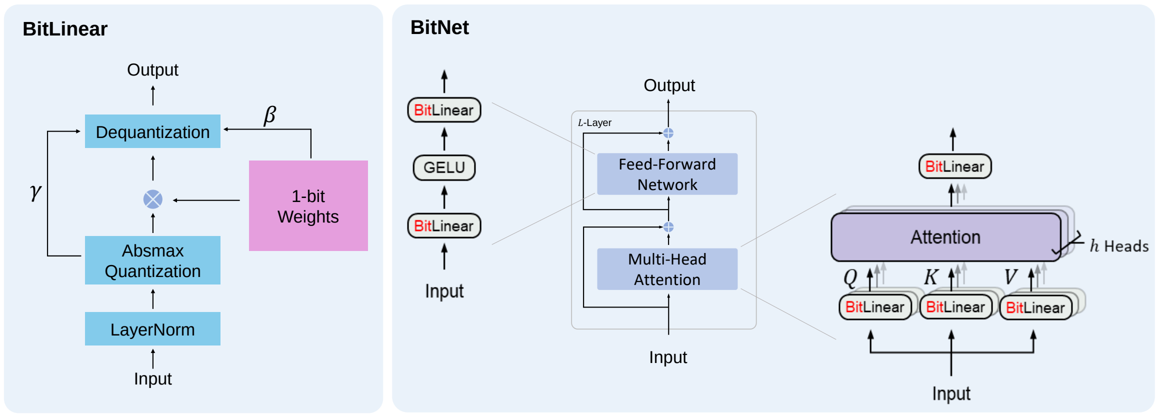

Figure 1: The overview of BitLinear and BitNet. **Left:** The computation flow of BitLinear. **Right:** An example of BitNet, consisting of the stacks of attentions and FFNs, where matrix multiplication is implemented as BitLinear. BitNet is easily applied for the other architectures (e.g., SwiGLU).

the same configuration (e.g., model size, training tokens, etc.). Besides, BitNet b1.58 is much more efficient in terms of memory consumption, throughput and latency compared to FP16 LLM baselines. More importantly, we show that BitNet follows a scaling law similar to that of half-precision Transformers, and the gap between FP16 LLM and BitNet becomes smaller as the model size grows, indicating that it can be effectively scaled to even larger language models with potential benefits in terms of performance and efficiency.

2. BitNet

As shown in Figure 1, BitNet uses the same layout as Transformers, stacking blocks of self-attention and feed-forward networks. Compared with vanilla Transformer, BitNet uses BitLinear (Eq. 9) instead of conventional matrix multiplication, which employs 1-bit or 1.58-bit model weights. We leave the other components high-precision, e.g., 8-bit in our experiments. We summarized the reasons as follows. First, the residual connections and the layer normalization contribute negligible computation costs to large language models. Second, the computation cost of QKV transformation is much smaller than the parametric projection as the model grows larger. Third, we preserve the precision for the input/output embedding because the language models have to use high-precision probabilities to perform sampling.

2.1 BitLinear

For BitNet b1 and BitNet b1.58, we first quantized the weights to $\{+1, -1\}$ and $\{+1, 0, -1\}$, respectively. For BitNet b1, we centralize the weights to be zero-mean before binarization to increase the capacity within a limited numerical range following Liu et al. (2022). A scaling factor β is used after binarization to reduce the $l2$ error between the real-valued and the binarized weights. The binarization of a weight $W \in \mathcal{R}^{n \times m}$ can be formulated as:

$$\widetilde{W} = \text{Sign}(W - \alpha), \tag{1}$$

$$\text{Sign}(W_{ij}) = \begin{cases} +1, & \text{if } W_{ij} > 0, \\ -1, & \text{if } W_{ij} \leq 0, \end{cases} \tag{2}$$

where α is $\frac{1}{nm}\sum_{ij} W_{ij}$. For BitNet b1.58, we adopt an *absmean* quantization function to constrain the weights to -1, 0, or +1. It first scales the weight matrix by its average absolute value, and then round each value to the nearest integer among {-1, 0, +1}:

$$\widetilde{W} = \text{RoundClip}(\frac{W}{\beta + \epsilon}, -1, 1), \tag{3}$$

$$\text{RoundClip}(x, a, b) = \max(a, \min(b, \text{round}(x))), \tag{4}$$

$$\beta = \frac{1}{nm}\sum_{ij} |W_{ij}|. \tag{5}$$

We further quantize the activations to b-bit (b=8) precision. Following Dettmers et al. (2022), we use absmax quantization, which scales activations into the range $[Q_n, Q_p]$ ($Q_n = -2^{b-1}$, $Q_p = 2^{b-1} - 1$) by multiplying with Q_p and dividing by the absolute maximum of the input matrix:

$$\widetilde{x} = \text{Quant}(x) = \text{RoundClip}(\frac{Q_p}{\gamma + \epsilon}x, Q_n, Q_p) \tag{6}$$

$$\text{where } \gamma = ||x||_\infty. \tag{7}$$

ϵ is a small floating-point number that prevents overflow when performing the clipping. Since the activations are more sensitive to the quantization, in this work, we quantize the activation to 8-bit and leave lower precision in future work. With the above quantization equations, the matrix multiplication can be written as $y = \widetilde{W}\widetilde{x}$. We assume that the elements in W and x are mutually independent and share the same distribution, and W and x are independent of each other. Then the variance of the output y is estimated as:

$$\text{Var}(y) = n\text{Var}(\widetilde{w}\widetilde{x}) = nE[\widetilde{w}^2]E[\widetilde{x}^2] = n\beta^2 E[\widetilde{x}^2] \approx E[\widetilde{x}^2] \tag{8}$$

For the half-precision computation, the variance of the output Var(y) is at the scale of 1 with the standard initialization methods (e.g., Kaiming initialization or Xavier initialization), which has a great benefit to the training stability. To preserve the variance after quantization, we introduce a normalization (Ba et al., 2016; Zhang and Sennrich, 2019) function before the activation quantization. In this way, the variance of the output y is then estimated as $\text{Var}(y) \approx E[\text{LN}(\widetilde{x})^2] = 1$, which has the same magnitude as the half-precision counterpart Var(y). In the context of Transformers, it has the exact implementation as `SubLN` (Wang et al., 2022). With `SubLN` and the quantization methods above, we have BitLinear, which is formulated as:

$$y = \widetilde{W}\widetilde{x} = \widetilde{W}\ \mathrm{Quant}(\mathrm{LN}(x)) \times \frac{\beta\gamma}{Q_p} \tag{9}$$

where β is $\frac{1}{nm}||W||_1$ for both BitNet b1 and BitNet b1.58. Figure 1 provides an illustration of the computation flow of BitLinear. After the SubLN operation, the activations are quantized with the absmax function. The matrix multiplication is performed between the 1-bit weights and the quantized activations. The output activations are rescaled with $\{\beta, \gamma\}$ to dequantize them to the original precision. For the model parallel (Shoeybi et al., 2019), we conduct the group quantization and normalization to accelerate the training. The weight and activations are normalized then quantized locally without requiring additional communication. More details can be found in the Appendix A.1. Furthermore, we also present the pseudocodes of BitNet b1.58 and BitNet b1 in the Appendix D and E, respectively.

2.2 Model Training

2.2.1 Straight-through estimator

To train our 1-bit model, we employ the straight-through estimator (STE)(Bengio et al., 2013) to approximate the gradient during backpropagation. This method bypasses the non-differentiable functions, such as the Sign (Eq. 2) and RoundClip (Eq. 4) functions, during the backward pass. STE allows gradients to flow through the network without being affected by these non-differentiable functions, making it possible to train our quantized model.

2.2.2 Mixed precision training

While the weights and the activations are quantized to low precision, the gradients and the optimizer states are stored in high precision to ensure training stability and accuracy. Following Liu et al. (2021), we maintain a latent weight in a high-precision format for the learnable parameters to accumulate the parameter updates. The latent weights are quantized on the fly during the forward pass and never used for the inference process.

2.2.3 Training dynamics

The loss curve during high-precision training typically exhibits an exponential shape. However, the curve observed during 1-bit training often follows an $S-$shaped pattern. As illustrated in the left part of Figure 2, there is a significant reduction in loss towards the end of the training process. A similar phenomenon has been observed in the training of binarized neural machine translation (Zhang et al., 2023). This suggests that intermediate evaluations may not accurately predict the final performance of 1-bit LLM training.

Besides, one challenge for the optimization is that a small update on the latent weights often makes no difference in the low-precision weights. This results in a biased gradient and update which are estimated based on the ultra low-bit weights. This problem is even worse at the beginning of the training, where the models are supposed to converge as fast as possible. To address this challenge, we explore various methods, concluding that increasing the learning rate is the simplest and best way to accelerate the optimization. Our initial

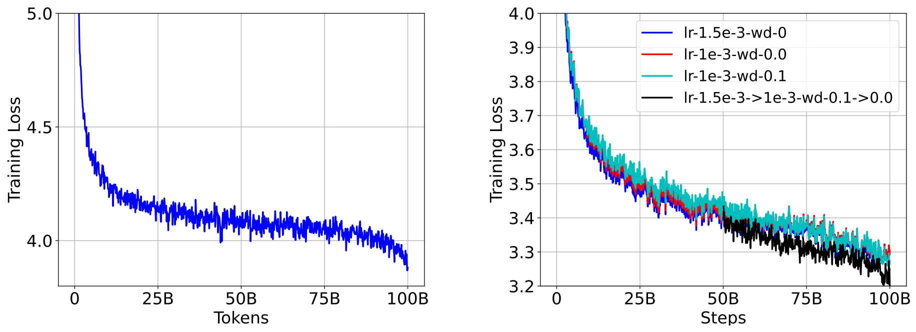

Figure 2: The training loss curves of BitNet b1.58. **Left:** The loss curve of 1.58-bit LLM is in $S-$shape. **Right:** The ablation of the learning rate and weight decay scheduling for BitNet.

trials revealed that half-precision models did not benefit from a similarly elevated learning rate. This is likely because the learning rates used for these models have already been well-tuned by prior work, and increasing them led to instability during the optimization process.

Halfway through the training process, we decayed the learning rate, a strategy that yielded better performance in our experiments. As illustrated in Figure 2, we observed a significant reduction in loss occurring when the learning rate was decayed, rather than at the end of the training phase. This approach made the performance more predictable during the training process. There has also been recent work on multi-step learning rate schedulers (Bi et al., 2024) for training language models, but these are primarily aimed at facilitating continual training rather than improving convergence.

In addition to the learning rate, weight decay also serves as a different role in 1-bit training. For half-precision training, weight decay acts as a regularizer, preventing over-fitting and improving training stability by constraining the magnitude of large weights. However, the effect of weight decay in 1-bit training differs substantially, as it is applied to the latent weights rather than the 1-bit weights themselves. In mixed-precision training, the magnitude of the latent weights can be interpreted as a confidence score for the corresponding 1-bit weights (Liu et al., 2021). Consequently, a large weight decay leads to lower confidence scores for the 1-bit weights, causing them to change more frequently. To mitigate this, we removed weight decay for the second half of the training, allowing the model to converge rather than update frequently. We empirically observed that after removing weight decay, the training loss decreased significantly faster, even without changing the learning rate. This weight decay scheduler is not applicable to half-precision models, as the role of weight decay fundamentally differs between half-precision and 1-bit learning.

Above all, we summarized our two-stage training recipe as follows. We present the learning rate and weight decay scheduling for the training of 700M models in Figure 10.

Bits	ADD Energy $\hat{E}_{add}$ (pJ) 45nm	7nm	MUL Energy $\hat{E}_{mul}$ (pJ) 45nm	7nm
FP32	0.9	0.38	3.7	1.31
FP16	0.4	0.16	1.1	0.34
INT8	0.03	0.007	0.2	0.07

Table 1: ADD and MUL energy consumption (Horowitz, 2014; Zhang et al., 2022) for different bit representations at 45nm and 7nm process nodes.

Models	**Size**	**WBits**	7nm Energy (J) MUL	ADD	45nm Energy (J) MUL	ADD
Transformer	6.7B	32	4.41	1.28	12.46	3.03
		16	1.14	0.54	3.70	1.35
BitNet		1	**0.02**	**0.04**	**0.08**	**0.13**
Transformer	13B	32	8.58	2.49	24.23	5.89
		16	2.23	1.05	7.20	2.62
BitNet		1	**0.04**	**0.06**	**0.12**	**0.24**
Transformer	30B	32	20.09	5.83	56.73	13.80
		16	5.21	2.45	16.87	6.13
BitNet		1	**0.06**	**0.14**	**0.20**	**0.53**

Table 2: Energy consumption of BitNet and Transformer varying different model size. Results are reported with 512 as input length. We estimate the energy of BitNet and Transformer according to the energy model in Zhang et al. (2022); Horowitz (2014).

1. In the first half of the training, we adopt a significantly higher learning rate and weight decay for Adam optimizer.
2. In the second half of the training, we decay the learning rate and disable the weight decay.

2.3 Arithmetic Operations Energy

We estimate the computational efficiency of BitNet in terms of arithmetic operations energy. We mainly focus on the calculation for the matrix multiplication, since it contributes the most to the cost of large language models.

According to the energy model in Horowitz (2014); Zhang et al. (2022), Table 1 demonstrates the energy consumption for different arithmetic operations. In vanilla Transformers, for matrix multiplication with the weight of dimensions $m \times n$ and the activations of $n \times p$, the energy consumption can be calculated as follows:

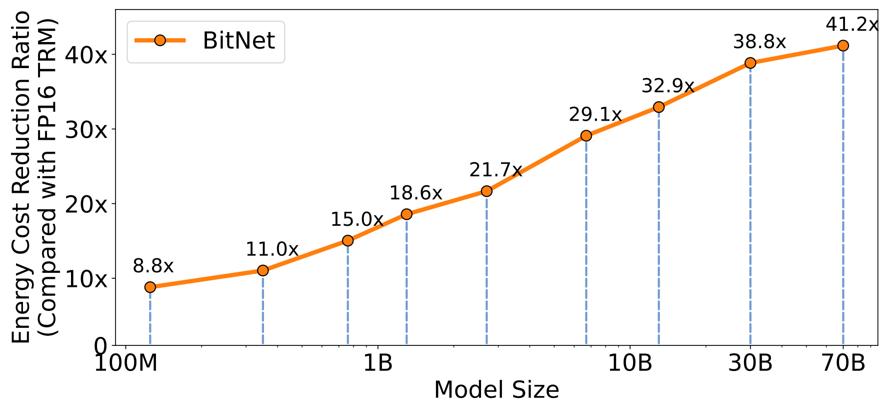

Figure 3: Energy consumption of BitNet compared to LLaMA LLM at 7nm process nodes across different model sizes.

$$E_{add} = m \times (n-1) \times p \times \hat{E}_{add} \tag{10}$$

$$E_{mul} = m \times n \times p \times \hat{E}_{mul} \tag{11}$$

For BitNet, the energy consumption of the matrix multiplication is dominated by the INT8 addition operations, as the weights are $\{-1,+1\}$ or $\{-1,0,+1\}$. The multiplication operations are only applied to scale the output with the scalars β and $\frac{\gamma}{Q_p}$, so the energy consumption for multiplication can be computed as:

$$E_{mul} = (m \times p + n \times p) \times \hat{E}_{mul} \tag{12}$$

which is significantly smaller than that in Transformers. The energy savings of BitNet compared to a full-precision (32-32) and half-precision (16-16) Transformer are shown in Table 2 and Figure 3. As can be seen, BitNet provides significant energy savings, especially for the multiplication operations, which are the major component of the matrix multiplication energy consumption. Figure 3 shows that as the model size scales, BitNet becomes increasingly more efficient in terms of energy consumption compared to the FP16 baseline. This is due to the fact that the percentage of *nn.Linear* grows with the model size, and the floating multiplication operations of BitNet is $\mathcal{O}(mp + np)$ which is one order of magnitude smaller than $\mathcal{O}(mnp)$ of FP16 LLM.

3. Experiments with BitNet b1

3.1 Comparison with FP16 Transformers

3.1.1 Setup

We train a series of autoregressive language models with BitNet b1 of various scales, ranging from 125M to 30B. The models are trained on an English-language corpus, which consists of

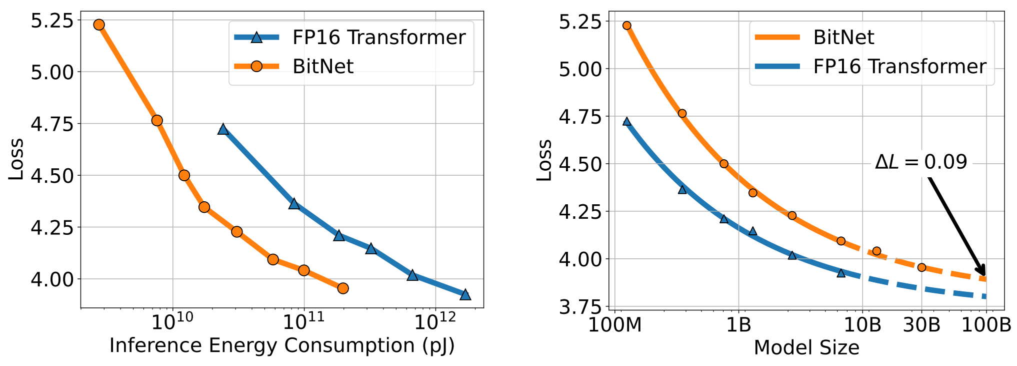

Figure 4: Scaling curves of BitNet b1 and FP16 Transformers.

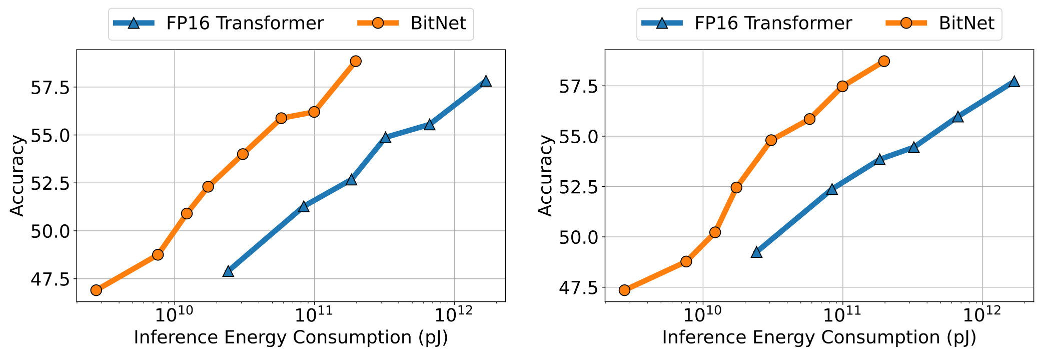

Figure 5: Zero-shot (Left) and few-shot (Right) performance of BitNet b1 and FP16 Transformer against the inference cost.

the Pile dataset, Common Crawl snapshots, RealNews, and CC-Stories datasets. We use the Sentencpiece tokenizer to preprocess data and the vocabulary size is 16K. For BitNet b1, we apply the unsigned absmax quantization after the non-linear activation function. The quantization for activations are performed per tensor during training and per token during inference. We deploy the first-stage training for all models. Besides, we also train the FP16 Transformer baselines with the same datasets and settings for a fair comparison. More details can be found in the Appendix B.

3.1.2 Inference-Optimal Scaling Law

Neural language models have proven to scale predictably (Kaplan et al., 2020) with vanilla Transformer architecture. The loss scales as the power law with the amount of computation used for training. This allows us to determine the optimal allocation of a computation budget as well as predict the performance of large language models from smaller models.

To study the scaling law of binarized Transformer, we start by plotting the scaling curve of both BitNet b1 and the FP16 Transformer baseline against the parameter count. We fix

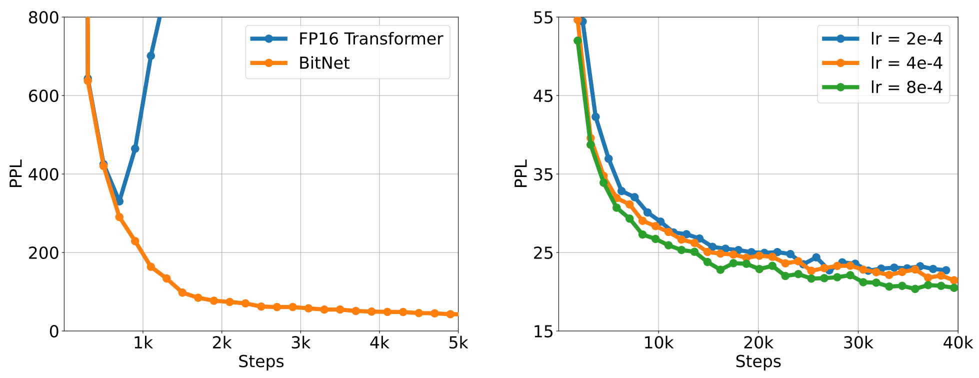

Figure 6: BitNet b1 is more stable than FP16 Transformer with a same learning rate (Left). The training stability enables BitNet b1 a larger learning rate, resulting in better convergence (Right).

the number of training tokens and vary the model sizes. Figure 4 shows that the loss scaling of BitNet b1 is similar to the FP16 Transformer, which follows a power-law. We then fit the scaling law with an irreducible loss term:

$$L(N) = aN^b + c \tag{13}$$

To evaluate whether the scaling law can accurately predict the loss, we choose the models from 125M to 6.7B to fit the parameters in the power-law and use the law to predict the loss of 13B and 30B. It shows that the fitted scaling law predicted BitNet b1's loss with high accuracy. More importantly, we observe that the gap between BitNet b1 and FP16 Transformer becomes smaller as the model size grows.

While the power-law above measures the trend of the scaling of BitNet b1, it does not properly model the relationship between the loss and the actual compute. Previous work (Kaplan et al., 2020; Henighan et al., 2020; Hoffmann et al., 2022) estimates the compute by calculating the FLOPs. However, it does not apply to 1-bit models whose cost is dominated by integer computation. Moreover, it mainly measures the training computation rather than the inference. To have a better understanding of the scaling efficiency of neural language models, we introduce Inference-Optimal Scaling Law. It predicts the loss against the energy consumption. We focus on the inference energy cost as it scales with the usage of the model, while the training cost is only once. We estimate the energy consumption as in Section 2.3. Figure 4 shows the scaling curve against the inference energy cost at 7nm process nodes. It proves that BitNet b1 has much higher scaling efficiency. Given a fixed computation budget, BitNet b1 achieves a significantly better loss. Meanwhile, the inference cost is much smaller to get the same performance as the FP16 models.

3.1.3 Results on Downstream Tasks

In addition to the loss, we are also concerned about the capabilities with the scaling of BitNet b1. Compared with the loss, the capacity is more difficult to predict due to the emergent nature of neural language models. To evaluate the capabilities with the interpretable metrics, we test both the 0-shot and 4-shot results on four downstream tasks, including Hellaswag (Zellers et al., 2019), Winogrande (Sakaguchi et al., 2020), Winograd (Levesque et al., 2012), and Storycloze (Mostafazadeh et al., 2016). Figure 5 reports the average results of BitNet b1 and FP16 Transformer with various scales. Similar to the loss scaling curve, the performance on the downstream tasks can scale as the computation budget grows. Besides, the scaling efficiency of capabilities is much higher than the FP16 Transformer baseline, in terms of both zero-shot and few-shot performance.

3.1.4 Stability Test

The major challenge for training ultra low-bit Transformers is the stability in optimization. Therefore, we perform stability tests for both BitNet b1 and the FP16 baseline by training a series of models with varying peak learning rates. Figure 6 illustrates the results of the stability test. It shows that BitNet b1 can converge with a large learning rate while FP16 Transformer can not, demonstrating better training stability of BitNet b1. This advantage in optimization enables the training with larger learning rates. It also demonstrates that BitNet b1 can benefit from the increase in learning rate, achieving better convergence in terms of PPL.

3.2 Comparison with Post-training Quantization

3.2.1 Setup

We train BitNet b1 with the same setup as described in Section 3.1.1. We compare BitNet b1 with state-of-the-art quantization methods, including Absmax (Dettmers et al., 2022), SmoothQuant (Xiao et al., 2023), GPTQ (Frantar et al., 2023), and QuIP (Chee et al., 2023). These methods are post-training quantization over an FP16 Transformer model, which follows the same training setting and data as BitNet b1. Among them, Absmax and SmoothQuant quantize both the weights and the activations, while GPTQ and QuIP only reduce the precision of weights. We apply the methods to various quantization levels. For the weight-only quantization (i.e., GPTQ and QuIP), we experiment with W4A16 and W2A16. For weight-and-activation quantization (i.e., Absmax and SmoothQuant), we use them to quantize the FP16 Transformers to W8A8, W4A4, and W1A8. Our implementation of BitNet b1 is binary weight 8-bit activation (W1A8), which has lower or equal bits than the baselines.

3.2.2 Results

Table 3 presents a detailed comparative analysis of the zero-shot performance of our proposed method, BitNet b1, against various baseline approaches on four benchmark datasets, namely Winogrande (Sakaguchi et al., 2020), Winograd (Levesque et al., 2012), Storycloze (Mostafazadeh et al., 2016), and Hellaswag (Zellers et al., 2019). All models have the model sizes of 6.7B for a fair comparison. The methods are evaluated across several weight bit levels, spanning from 16 down to 1. Besides the zero-shot accuracy on the downstream tasks, the evaluation metrics include language model perplexity on the validation set, which provides a comprehensive understanding of each method's performance.

WBits	**Methods**	**PTQ**	**PPL↓**	**WG↑**	**WGe↑**	**HS↑**	**SC↑**	**Avg.↑**
-	Random	-	-	50.0	50.0	25.0	50.0	43.8
16	Transformer	✗	15.19	66.7	54.3	42.9	67.4	57.8
8	Absmax	✓	21.43	60.4	52.0	38.3	62.7	53.4
	SmoothQuant	✓	15.67	65.3	53.1	40.9	67.6	56.7
4	GPTQ	✓	16.05	57.2	51.2	39.9	63.4	52.9
	Absmax	✓	4.8e4	55.8	50.9	25.0	53.1	46.2
	SmoothQuant	✓	1.6e6	53.7	48.3	24.8	53.6	45.1
2	GPTQ	✓	1032	51.6	50.1	25.8	53.4	45.2
	QuIP	✓	70.43	56.1	51.2	30.3	58.4	49.0
1	Absmax	✓	3.5e23	49.8	50.0	24.8	53.6	44.6
	SmoothQuant	✓	3.3e21	50.5	49.5	24.6	53.1	44.4
1	**BitNet b1**	✗	17.07	66.3	51.4	38.9	66.9	55.9

Table 3: Zero-shot results for BitNet b1 and the baselines (`PTQ`: Post-training quantization).

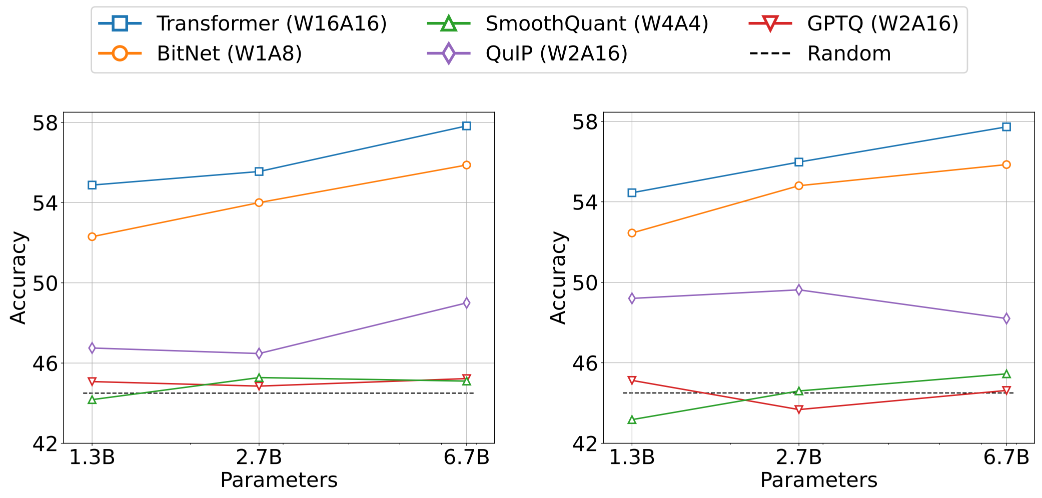

Figure 7: Zero-shot (Left) and few-shot (Right) results for BitNet b1 and the post-training quantization baselines on downstream tasks. We report the average accuracy on the Winogrande, Winograd, Storycloze and Hellaswag dataset.

The results demonstrate the effectiveness of BitNet b1 in achieving competitive performance levels compared to the baseline approaches, particularly for lower bit levels. The zero-shot scores of BitNet b1 are comparable with the 8-bit models, while the inference cost is much lower. For the 4-bit models, the weight-only quantization methods outperform the weight-and-activation quantizers, mainly because the activation is more difficult to

Models	**Size**	**Memory (GB)↓**	**Latency (ms)↓**	**PPL↓**
LLaMA LLM	700M	2.08 (1.00x)	1.18 (1.00x)	12.33
BitNet b1.58	700M	0.80 (2.60x)	0.96 (1.23x)	12.87
LLaMA LLM	1.3B	3.34 (1.00x)	1.62 (1.00x)	11.25
BitNet b1.58	1.3B	1.14 (2.93x)	0.97 (1.67x)	11.29
LLaMA LLM	3B	7.89 (1.00x)	5.07 (1.00x)	10.04
BitNet b1.58	3B	**2.22 (3.55x)**	**1.87 (2.71x)**	**9.91**
BitNet b1.58	3.9B	**2.38 (3.32x)**	**2.11 (2.40x)**	**9.62**

Table 4: Perplexity as well as the cost of BitNet b1.58 and LLaMA LLM.

Models	**Size**	**ARCe**	**ARCc**	**HS**	**BQ**	**OQ**	**PQ**	**WGe**	**Avg.**
LLaMA LLM	700M	54.7	23.0	37.0	60.0	20.2	68.9	54.8	45.5
BitNet b1.58	700M	51.8	21.4	35.1	58.2	20.0	68.1	55.2	44.3
LLaMA LLM	1.3B	56.9	23.5	38.5	59.1	21.6	70.0	53.9	46.2
BitNet b1.58	1.3B	54.9	24.2	37.7	56.7	19.6	68.8	55.8	45.4
LLaMA LLM	3B	62.1	25.6	43.3	61.8	24.6	72.1	58.2	49.7
BitNet b1.58	3B	**61.4**	**28.3**	**42.9**	**61.5**	**26.6**	**71.5**	**59.3**	**50.2**
BitNet b1.58	3.9B	**64.2**	**28.7**	**44.2**	**63.5**	**24.2**	**73.2**	**60.5**	**51.2**

Table 5: Zero-shot accuracy of BitNet b1.58 and LLaMA LLM on the end tasks.

quantify. BitNet b1, as a 1-bit model, significantly achieves better results than both the weight-and-activation quantization methods and the weight-only quantization methods.

As for the lower-bit models, BitNet b1 has consistently superior scores over all baselines. This proves the advantages of the quantization-aware training approaches over the post-training quantization methods. Figure 7 summarizes both the zero-shot accuracy and few-shot accuracy of our method and the baselines while scaling up the model size from 1.3B to 6.7B. It proves that the advantage is consistent across different scales.

4. Experiments with BitNet b1.58

4.1 Setup

We compared BitNet b1.58 to our reproduced FP16 LLaMA LLM in various sizes. For BitNet b1.58, we adopt the LLaMA-alike components, including RMSNorm (Zhang and Sennrich, 2019), SwiGLU (Shazeer, 2020), rotary embedding (Su et al., 2024), and removing all biases. The activations are all scaled to $[-Q_b, Q_b]$ per token to get rid of the zero-point quantization. This is more convenient and simple for both implementation and system-level optimization, while introduces negligible effects to the performance in our experiments. To ensure a fair comparison, we pre-trained the models on the RedPajama dataset (Computer, 2023) for 100 billion tokens. We present the training loss curves for BitNet b1.58 and the baselines in the Appendix A.2. More details about the hyper-parameters are summarized in the Appendix C.

Models	**Size**	**ARCe**	**ARCc**	**HS**	**BQ**	**OQ**	**PQ**	**WGe**	**Avg.**
BitNet b1.58	3B	61.4	28.3	42.9	61.5	26.6	71.5	59.3	50.2
w/ 4-bit KV	3B	60.9	27.3	42.9	61.4	26.2	71.6	59.9	50.0
BitNet b1.58	3.9B	64.2	28.7	44.2	63.5	24.2	73.2	60.5	51.2
w/ 4-bit KV	3.9B	64.1	28.8	44.1	63.7	24.2	73.2	61.1	51.3

Table 6: Zero-shot accuracy on the benchmark. The KV cache of BitNet can be losslessly quantized to 4 bits after training.

4.2 Results

We evaluated the zero-shot performance on a range of language tasks, including ARC-Easy (Yadav et al., 2019), ARC-Challenge (Yadav et al., 2019), Hellaswag (Zellers et al., 2019), Winogrande (Sakaguchi et al., 2020), PIQA (Bisk et al., 2019), OpenbookQA (Mihaylov et al., 2018), and BoolQ (Clark et al., 2019). We also reported the average validation perplexity on the WikiText2 (Merity et al., 2016) and C4 (Raffel et al., 2019) datasets.

We compared the runtime GPU memory and latency of both LLaMA LLM and BitNet b1.58. The results were measured using the FasterTransformer[1] codebase, which is well-optimized for LLM inference latency on GPU devices. The 2-bit kernel from Ladder (Wang et al., 2023) is also integrated for BitNet b1.58. We reported the time per output token, as it is the major cost for inference.

Table 4 summarizes the perplexity and the cost for BitNet b1.58 and LLaMA LLM. It shows that BitNet b1.58 starts to match half-precision LLaMA LLM at 3B model size in terms of perplexity, while being 2.71 times faster and using 3.55 times less GPU memory. In particular, BitNet b1.58 with a 3.9B model size is 2.4 times faster, consumes 3.32 times less memory, but performs significantly better than LLaMA LLM 3B.

Table 5 reports the detailed results of the zero-shot accuracy on the end tasks. We followed the pipeline from *lm-evaluation-harness*[2] to perform the evaluation. The results show that the performance gap between BitNet b1.58 and LLaMA LLM narrows as the model size increases. More importantly, BitNet b1.58 can match the performance of the half-precision baseline starting from a 3B size. Similar to the observation of the perplexity, the end-task results reveal that BitNet b1.58 3.9B outperforms LLaMA LLM 3B with lower memory and latency cost. This demonstrates that BitNet b1.58 is a Pareto improvement over the state-of-the-art LLM models.

Low-bit attention is critical for efficient long-context modeling, as it significantly reduces the memory footprint of KV cache which increases linearly as the input's length becomes longer. We quantize each head using symmetric absmax function. As shown in Table 6, the KV cache of BitNet b1.58 can be compressed to 4-bit integers without any accuracy loss in 3B and 3.9B model sizes.

1. `https://github.com/NVIDIA/FasterTransformer`
2. `https://github.com/EleutherAI/lm-evaluation-harness`

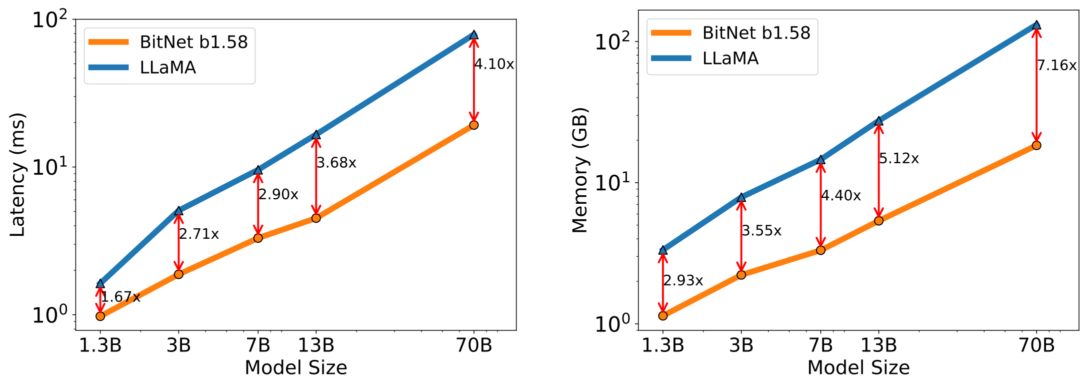

Figure 8: Decoding latency (Left) and memory consumption (Right) of BitNet b1.58 varying the model size.

Models	Size	Max Batch Size	Throughput (tokens/s)
LLaMA LLM	70B	16 (1.0x)	333 (1.0x)
BitNet b1.58	70B	**176 (11.0x)**	**2977 (8.9x)**

Table 7: Comparison of the throughput between BitNet b1.58 70B and LLaMA LLM 70B.

4.3 Inference Cost

In this section, we evaluate the inference cost of BitNet b1.58 and FP16 LLM on the latency, memory and throughput. BitNet b1.58 is enabling a new scaling law with respect to model performance and inference cost. As a reference, we can have the following equivalence between different model sizes in 1.58-bit and 16-bit based on the results in Figure 8.

- 13B BitNet b1.58 is more efficient, in terms of latency, memory usage and energy consumption, than 3B FP16 LLM.
- 30B BitNet b1.58 is more efficient, in terms of latency, memory usage and energy consumption, than 7B FP16 LLM.
- 70B BitNet b1.58 is more efficient, in terms of latency, memory usage and energy consumption, than 13B FP16 LLM.

Memory and Latency. We further scaled up the model size to 7B, 13B, and 70B and evaluated the cost. Figure 8 illustrates the trends of latency and memory, showing that the speed-up increases as the model size scales. In particular, BitNet b1.58 70B is 4.1 times faster than the LLaMA LLM baseline. This is because the time cost for *nn.Linear* grows with the model size. The memory consumption follows a similar trend, as the embedding remains half-precision and its memory proportion is smaller for larger models. Both latency

Models	**Tokens**	**WGe**	**PIQA**	**SciQ**	**LAMBADA**	**ARC-easy**	**Avg.**
StableLM-3B	2T	64.56	76.93	90.75	66.09	67.78	73.22
BitNet b1.58 3B	2T	**66.37**	**78.40**	**91.20**	**67.63**	**68.12**	**74.34**

Table 8: Comparison of BitNet b1.58 with StableLM-3B with 2T tokens.

and memory were measured with a 2-bit kernel, so there is still room for optimization to further reduce the cost.

Throughput. We compare the throughput of BitNet b1.58 and LLaMA LLM with 70B parameters on two 80GB A100 cards, using pipeline parallelism (Huang et al., 2019) so that LLaMA LLM 70B could be run on the devices. We increased the batch size until the GPU memory limit was reached, with a sequence length of 512. Table 7 shows that BitNet b1.58 70B can support up to 11 times the batch size of LLaMA LLM, resulting an 8.9 times higher throughput.

4.4 Training with 2T Tokens

The number of training tokens is a crucial factor for LLMs. To test the scalability of BitNet b1.58 in terms of tokens, we trained a BitNet b1.58 model with 2T tokens following the data recipe of StableLM-3B (Tow et al.), which is the state-of-the-art open-source 3B model. Both models were evaluated on a benchmark that consists of Winogrande (Sakaguchi et al., 2020), PIQA (Bisk et al., 2019), SciQ (Welbl et al., 2017), LAMBADA (Paperno et al., 2016), and ARC-easy (Yadav et al., 2019). We reported the zero-shot accuracy in Table 8. For tasks measured with accuracy and normalized accuracy, we take the average of the two. The results of StableLM 3b at 2T tokens are taken directly from its technical report. Our findings shows that BitNet b1.58 achieves a superior performance on all end tasks, indicating that 1.58-bit LLMs also have strong generalization capabilities.

5. Ablation Studies

In Table 9, we present an ablation study of BitNet compared with several alternative approaches. We ablate the effect of our choices in activation quantization approaches as well as the techniques to stabilize the model training. To ensure a fair comparison, all models are trained with 1-bit weight and the same data from scratch. BitNet implements absmax to quantize the activation and use `SubLN` for training stability. One quantization alternative is the elastic function (Liu et al., 2022), which dynamically adjusts the scales with learnable parameters. In our experiments, we find that absmax has better performance than the elastic function. Besides, the absmax function leads to more stable training, which enables a larger learning rate for BitNet. We further compare `SubLN` with the Pre-LN and the BMT architecture (Zhang et al., 2023). Pre-LN is the default architecture for GPT pertaining, while BMT has proven to improve the stability of binarized models. Our experiments show that `SubLN` outperforms both Pre-LN and BMT. Therefore, we choose absmax and `SubLN` as the implementation in BitNet.

Methods	PPL↓	HS↑	WGe↑	WG↑	SC↑	Avg.↑
Zero-Shot Learning						
BitNet	**20.34**	33.2	52.1	60.7	63.2	**52.3**
Elastic + Pre-LN	24.05	29.6	52.9	56.8	61.3	50.2
Absmax + Pre-LN	22.11	31.6	50.0	61.8	61.6	51.3
Absmax + BMT	22.98	31.2	52.1	60.4	62.7	51.6
Few-Shot Learning						
BitNet	**20.34**	33.5	50.4	62.1	63.8	**52.5**
Elastic + Pre-LN	24.05	29.9	51.7	57.5	61.1	50.1
Absmax + Pre-LN	22.11	31.4	51.9	63.9	61.6	52.2
Absmax + BMT	22.98	31.3	51.5	57.5	62.6	50.7

Table 9: Ablation of BitNet. Elastic is an activation quantization method from Liu et al. (2022), while BMT is the architecture from Zhang et al. (2023) to stabilize the training of low-bit models. All models are trained with 1-bit weight from scratch.

6. Related Work

Recent years have seen a rapid growth in the size and capabilities of LLMs. These models have shown remarkable performance in a wide scope of tasks, but their increasing size has posed challenges for deployment. Model quantization has emerged as a promising solution, which reduces the precision of weights and activations, significantly reducing the memory and computational requirements of LLMs.

Most existing quantization methods for large language models are post-training (Dettmers et al., 2022; Frantar et al., 2023; Lin et al., 2023; Chee et al., 2023; Tseng et al., 2024; Shang et al., 2023; Shao et al., 2023). These methods can be divided into two categories: weight-only and weight-activation quantization. As for the weight-activations quantization, Dettmers et al. (2022) adopts a mixed-precision decomposition: it isolates the outlier feature dimensions into a 16-bit matrix multiplication, and the rest majority are vector-wise quantized into 8-bit to perform matrix multiplication. Further, Xiao et al. (2023) proposed SmoothQuant to first smooth the activation outliers by offline migrating the quantization error from activations to weights, then use Absmax to quantize the weights and activations into 8-bit. For the weight-only quantization, GPTQ (Frantar et al., 2023) and AWQ (Lin et al., 2023) can reduce the bit-width down to 4 bits per weight, with competitive performance relative to the uncompressed baseline. QuIP# (Tseng et al., 2024) uses the improved incoherence processing from QuIP (Chee et al., 2023) and vector quantization in incoherent weights possess. With the additional fine-tuning, QuIP# achieved state-of-the-art results in 2-bit models. However, it still suffers from the degradation of accuracy, and has more computation cost and lower inference speed.

Another line of research lies on the quantization-aware training (Peng et al., 2023; Liu et al., 2023b). Compared with post-training methods, they usually introduce more training cost but have better performance, since these quantized models are trained from the scratch. Peng et al. (2023) proposed automatic mixed-precision framework for training FP8 LLMs.

Liu et al. (2023b) introduced LLM-FP4 for quantizing both weights and activations down to 4-bit for the continue-training of LLMs.

7. Discussion

BitNet has demonstrated the surprising effectiveness of pre-training with 1-bit and 1.58-bit weight for LLMs. Extensive experiments reveal that as model size increases, the performance gap between BitNet and FP16 LLMs narrows. This trend is also observed in post-training quantization methods (Frantar et al., 2023) for 4-bit LLMs, suggesting that LLMs exhibit increased parameter redundancy as their total size grows. Recent works (Daliri et al., 2024) also theoretically proved that the dynamics of 1-bit model training align with kernel behavior, and the generalization difference between 1-bit networks and their half-precision counterparts maintains a negligible level as the network width increases. Consequently, ultra low-precision models can be directly trained at larger sizes without sacrificing performance. As illustrated in Figure 4, the validation loss gap between BitNet b1 and FP16 Transformer decreases from 0.5 to 0.09 as model sizes increase from 125 million to 100 billion parameters, indicating the potential to train binarized LLMs with trillions of parameters without performance degradation.

Ternarization is a notable variant of binarization, offering enhanced modeling capability through explicit feature filtering. Bai et al. showed that the loss surface of ternary BERT is locally convex and easier to optimize than that of the binary model. BitNet b1.58 with ternary weights retains all the advantages of BitNet b1, including the novel computation paradigm that minimizes multiplication operations for matrix multiplication to reduce the energy consumption and allows for high optimization. Crucially, our experiments demonstrate that BitNet b1.58 can achieve FP16 baseline performance with the same configuration, starting from just 3 billion parameters—substantially fewer than required for BitNet b1.

8. Conclusion and Future Work

We present BitNet, a novel 1-bit Transformer architecture for large language models. Our approach is designed to be scalable and stable, with the ability to handle large language models efficiently. Extensive experimental results demonstrate that BitNet b1 achieves competitive performance in terms of both perplexity and downstream task performance, while significantly reducing memory footprint and energy consumption compared to the baselines. Moreover, we observe that the gap between BitNet and FP16 baseline narrows as the model size grows, which indicates that for a large-scale model, we can easily quantize it into a ultra low-bit model during the pre-training without suffering from performance degradation. The experiments show that BitNet b1.58 with the ternary weight matches half-precision (i.e., FP16 or BF16) baselines in terms of both perplexity and end-task performance, starting from a 3B size, when using the same configuration (e.g., model size, training tokens, etc.). In the future, we would like to scale up BitNet in terms of model size and training steps. We are also interested in applying BitNet in other architectures (e.g., RetNet, Sun et al. (2023)) for training large language models.

9. Limitations

Our work empirically demonstrates the effectiveness of 1-bit pre-training for LLMs and reveals several interesting phenomena observed during training. We discuss related theoretical justifications in Section 7, while a more comprehensive theoretical analysis is left for future work. Additionally, our current evaluation focuses primarily on English language understanding and reasoning tasks using a decoder-only architecture. As part of our future work, we plan to extend 1-bit pre-training to encoder-decoder architectures, multilingual benchmarks, and more complex tasks such as mathematical reasoning and code generation.

Appendix A. Training Details

A.1 Model parallelism with Group Quantization and Normalization.

One essential technique to scale up large language models is model parallelism (Shoeybi et al., 2019), which partitions the matrix multiplication on multiple devices. A prerequisite for the existing model parallelism approaches is that the tensors are independent along the partition dimension. However, all of the parameters α, β, γ, and η are calculated from the whole tensors, breaking the independent prerequisite. One solution is to introduce one *all-reduce* operation for each parameter. However, even though the communication for each parameter is small, the amount of synchronization is growing as the model becomes deeper, which significantly slows the forward pass. The problem also exists in `SubLN`, where the mean and the variance should be estimated across the partition dimension.

To this end, we propose a simple approach that makes the model parallelism more efficient. We divide the weights and activations into groups and then independently estimate each group's parameters. This way, the parameters can be calculated locally without requiring additional communication. This approach, called Group Quantization, is formulated as follows.

For a weight matrix $W \in \mathcal{R}^{n\times m}$, we divide it into G groups along the partition dimension, and each group has a size of $\frac{n}{G} \times m$. We then estimate the parameters for each group independently:

$$\alpha_g = \frac{G}{nm}\sum_{ij} W^{(g)}_{ij}, \quad \beta_g = \frac{G}{nm}||W^{(g)}||_1, \tag{14}$$

where $W^{(g)}$ denotes the g-th group of the weight matrix. Similarly, for the activations, we can divide the input matrix $x \in \mathcal{R}^{n\times m}$ into G groups and calculate the parameters for each group:

$$\gamma_g = ||x^{(g)}||_\infty, \quad \eta_g = \min_{ij} x^{(g)}_{ij} \tag{15}$$

For LN, we can apply the group normalization technique (Wu and He, 2018) to compute the mean and variance for each group independently. In this way, we can efficiently implement model parallelism with Group Quantization and Normalization, which requires no additional communication and can scale to large language models.

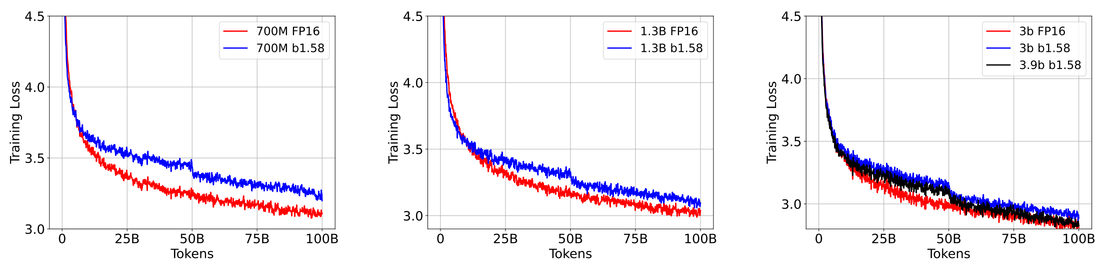

Figure 9: Training loss curves across different model sizes. The gap between half-precision models and BitNet b1.58 becomes narrower as the models scale.

A.2 The training curves of BitNet b1.58

We present the training loss curves for BitNet b1.58 and the baselines in this section. As shown in Figure 9, these curves illustrate that the gap between the half-precision LLaMA LLM and BitNet b1.58 diminishes as the model size increases, indicating that 1-bit language models perform better with larger model size. Interestingly, while there is no discernible gap between the 3B versions of BitNet b1.58 and LLaMA LLM in terms of validation loss and end-task accuracy, a slight difference exists in the training loss. This suggests that 1.58-bit models may exhibit better generalization capabilities and be less prone to overfitting. We leave a more in-depth analysis of this phenomenon for future work.

Appendix B. Hyperparameters on BitNet b1

Table 10 presents the configuration and hyper-parameters for BitNet b1 in the scaling experiments in each model size. The dropout and gradient clipping are disabled during pre-training. For 13B and 30B model, we set weight decay to 0.05 for training stability. As for the stability test of BitNet b1 and FP16 Transformer, we adopt a higher learning rate and small batch size. The peak learning rate is set as 1e-3, and the batch size is 64. For all experiments, the sequence length is set as 2,048 tokens.

Size	**# Hidden**	**# Layers**	**# Heads**	**LR**	**Weight decay**	**Steps**	**Adam β**	**Warmup**
125M	768	12	12	2.4e-3	0.01	40K	(0.9, 0.98)	750
350M	1024	24	16	1.2e-3	0.01	40K	(0.9, 0.98)	750
760M	1536	24	16	1e-3	0.01	40K	(0.9, 0.98)	750
1.3B	2048	24	32	8e-4	0.01	40K	(0.9, 0.98)	750
2.7B	2560	32	32	6.4e-4	0.01	40K	(0.9, 0.98)	750
6.7B	4096	32	32	4.8e-4	0.01	40K	(0.9, 0.98)	750
13B	5120	40	40	4e-4	0.05	40K	(0.9, 0.98)	750
30B	7168	48	56	4e-4	0.05	40K	(0.9, 0.98)	750

Table 10: Model configuration and hyper-parameters for BitNet b1 in the scaling experiments. For 13B and 30B model, we set weight decay to 0.05 for training stability. We adopt polynomial decay as learning rate scheduler. The batch size is 128.

Hyperparameters	Elastic	Absmax
Peak learning rate	1e-4	8e-4
Training updates	40K	
Tokens per sample	256K	
Adam β	(0.9, 0.98)	
Learning rate scheduler	Polynomial decay	
Warmup updates	750	
Weight decay	0.01	

Table 11: Hyperparameters for the ablations of BitNet b1.

Appendix C. Hyperparameters on BitNet b1.58

We report the configurations and hyper-parameters for BitNet b1.58 in Table 12 and Table 13, respectively. For all experiments, the sequence length is set as 2,048 tokens. The batch size is 512, resulting in up to 1M tokens. The Adam β is set as (0.9, 0.95). Figure 10 illustrates the learning rate and weight decay scheduler in the training of 700M BitNet b1.58.

Size	Hidden Size	GLU Size	#Heads	#Layers
700M	1536	4096	24	24
1.3B	2048	5460	32	24
3B	3200	8640	32	26
3.9B	3200	12800	32	26

Table 12: Model configurations for both BitNet b1.58 and LLaMA LLM.

Model	Size	Learning Rate	Weight Decay	Batch Size	Steps	Warmup
BitNet b1.58	700M	$1.5 \times 10^{-3} \rightarrow 1 \times 10^{-3}$	$0.1 \rightarrow 0$	1M	100K	375
	1.3B	$1.2 \times 10^{-3} \rightarrow 8 \times 10^{-4}$	$0.1 \rightarrow 0$	1M	100K	375
	3B	$1.2 \times 10^{-3} \rightarrow 8 \times 10^{-4}$	$0.1 \rightarrow 0$	1M	100K	375
	3.9B	$1.2 \times 10^{-3} \rightarrow 8 \times 10^{-4}$	$0.1 \rightarrow 0$	1M	100K	375
LLaMA LLM	700M	2.5×10^{-4}	0.1	1M	100K	375
	1.3B	2.0×10^{-4}	0.1	1M	100K	375
	3B	2.0×10^{-4}	0.1	1M	100K	375

Table 13: Hyper-parameters for both BitNet b1.58 and LLaMA LLM training.

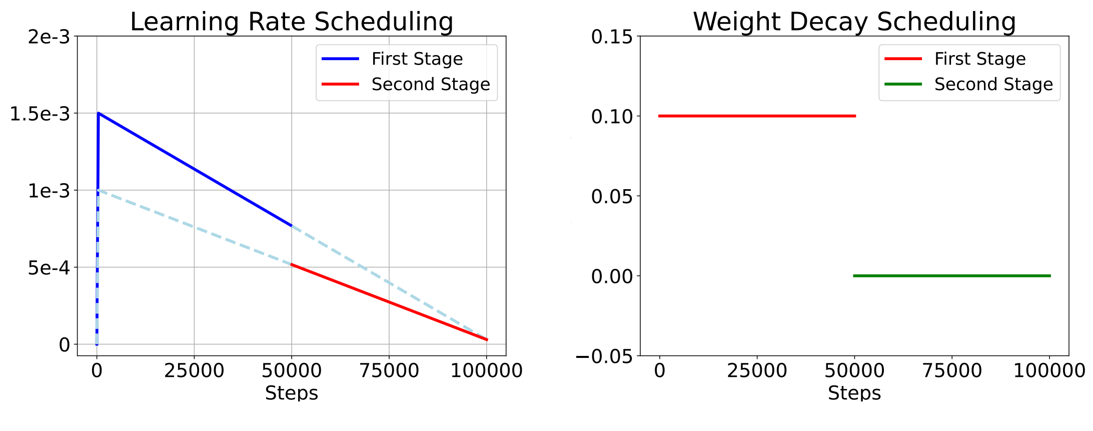

Figure 10: The learning rate and the weight decay scheduler in the training of 700M BitNet b1.58.

Appendix D. PyTorch Implementation of BitNet b1.58

There are two steps to change from a LLaMA LLM architecture to BitNet b1.58:

1. Replace all *nn.Linear* in attention and SwiGLU with *BitLinear* (Figure 11);
2. Remove RMSNorm before attention and SwiGLU because *BitLinear* has built-in RMSNorm.

```
def activation_quant(x):
""" Per-token quantization to 8 bits. No grouping is needed for quantization.
Args:
  x: an activation tensor with shape [n, d]
Returns:
  y: a quantized activation tensor with shape [n, d]
"""
  scale = 127.0 / x.abs().max(dim=-1, keepdim=True).values.clamp_(min=1e-5)
  y = (x * scale).round().clamp_(-128, 127) / scale
  return y

def weight_quant(w):
""" Per-tensor quantization to 1.58 bits. No grouping is needed for quantization.
Args:
  w: a weight tensor with shape [d, k]
Returns:
  u: a quantized weight with shape [d, k]
"""
  scale = 1.0 / w.abs().mean().clamp_(min=1e-5)
  u = (w * scale).round().clamp_(-1, 1) / scale
  return u

class BitLinear(nn.Linear):
"""
This is only for training, and kernel optimization is needed for efficiency.
"""
  def forward(self, x):
  """
  Args:
    x: an input tensor with shape [n, d]
  Returns:
    y: an output tensor with shape [n, d]
  """
    w = self.weight # a weight tensor with shape [d, k]
    x_norm = RMSNorm(x)
    # A trick for implementing Straight-Through-Estimator (STE) using detach()
    x_quant = x_norm + (activation_quant(x_norm) - x_norm).detach()
    w_quant = w + (weight_quant(w) - w).detach()
    y = F.linear(x_quant, w_quant)
    return y
```

Figure 11: Pytorch code for the BitLinear component in training BitNet. It requires additional efforts to improve the training efficiency, such as kernel fusion.

As for the inference, there are some changes for efficiency.

1. The model weights are offline quantized to 1.58 bits.
2. The standard *F.linear* operation is replaced with a customized low-bit kernel.
3. The scaling factors for both weight quantization and activation quantization are applied after the *F.linear* operation.
4. There is no need to implement the Straight-Through Estimator (STE) trick.

5. The RMSNorm operation can be fused with the activation quantization.

```
def activation_norm_quant(x):
""" RMSNorm & Per−token quantization to 8 bits. It can be implemented as a fused kernel.
Args:
  x: an activation tensor with shape [n, d]
Returns:
  y: a quantized activation tensor with shape [n, d]
  scale: a scalar for dequantization with shape [1]
"""
  x = RMSNorm(x)
  scale = 127.0 / x.abs().max(dim=−1, keepdim=True).values.clamp_(min=1e−5)
  y = (x ∗ scale).round().clamp_(−128, 127)
  return y, scale

class BitLinear(nn.Linear):
"""
This is only for inference. The weights should been quantized in advance.
"""
  def forward(self, x):
  """
  Args:
    x: an input tensor with shape [n, d]
  Returns:
    y: an output tensor with shape [n, d]
  """
    w = self.weight # a 1.58−bit weight tensor with shape [d, k]
    w_scale = self.weight_scale # a half−precision weight scale tensor with shape [1]
    x_quant, x_scale = activation_norm_quant(x)
    y = gemm_lowbit_kernel(x_quant, w) / w_scale / x_scale
    return y
```

Figure 12: Pytorch code for the BitLinear component during inference.

Appendix E. PyTorch Implementation of BitNet b1

Similarly, we provide the implementation of the original 1-bit BitNet by replacing the function of *weight_quant(w)* as in Figure 11.

```
def weight_quant(w):
""" Per−tensor quantization to 1 bits. No grouping is needed for quantization.
Args:
  w: a weight tensor with shape [d, k]
Returns:
  u: a quantized weight with shape [d, k]
"""
  scale = w.abs().mean()
  e = w.mean()
  u = (w − e).sign() ∗ scale
  return u
```

Figure 13: Pytorch code for the BitLinear component in BitNet b1.

References

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, and et al. PaLM 2 technical report. *CoRR*, abs/2305.10403, 2023.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, 2016.

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jin Jin, Xin Jiang, Qun Liu, Michael R. Lyu, and Irwin King. Binarybert: Pushing the limit of BERT quantization. In *ACL/IJCNLP 2021*, pages 4334–4348. Association for Computational Linguistics.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, Kaige Gao, Wenjun Gao, Ruiqi Ge, Kang Guan, Daya Guo, Jianzhong Guo, Guangbo Hao, Zhewen Hao, Ying He, Wenjie Hu, Panpan Huang, Erhang Li, Guowei Li, Jiashi Li, Yao Li, Y. K. Li, Wenfeng Liang, Fangyun Lin, Alex X. Liu, Bo Liu, Wen Liu, Xiaodong Liu, Xin Liu, Yiyuan Liu, Haoyu Lu, Shanghao Lu, Fuli Luo, Shirong Ma, Xiaotao Nie, Tian Pei, Yishi Piao, Junjie Qiu, Hui Qu, Tongzheng Ren, Zehui Ren, Chong Ruan, Zhangli Sha, Zhihong Shao, Junxiao Song, Xuecheng Su, Jingxiang Sun, Yaofeng Sun, Minghui Tang, Bingxuan Wang, Peiyi Wang, Shiyu Wang, Yaohui Wang, Yongji Wang, Tong Wu, Y. Wu, Xin Xie, Zhenda Xie, Ziwei Xie, Yiliang Xiong, Hanwei Xu, R. X. Xu, Yanhong Xu, Dejian Yang, Yuxiang You, Shuiping Yu, Xingkai Yu, B. Zhang, Haowei Zhang, Lecong Zhang, Liyue Zhang, Mingchuan Zhang, Minghua Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, Qihao Zhu, and Yuheng Zou. Deepseek LLM: scaling open-source language models with longtermism. *CoRR*, abs/2401.02954, 2024.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. *CoRR*, abs/1911.11641, 2019.

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, and et al. Language models are few-shot learners. In *NeurIPS*, 2020.

Adrian Bulat and Georgios Tzimiropoulos. XNOR-Net++: improved binary neural networks. In *BMVC*, 2019.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. QuIP: 2-bit quantization of large language models with guarantees. *CoRR*, abs/2307.13304, 2023.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, and et al. PaLM: scaling language modeling with pathways. *CoRR*, abs/2204.02311, 2022.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *CoRR*, abs/1905.10044, 2019.

Together Computer. Redpajama: an open dataset for training large language models, 2023. URL `https://github.com/togethercomputer/RedPajama-Data`.

Majid Daliri, Zhao Song, and Chiwun Yang. Unlocking the theory behind scaling 1-bit neural networks. *arXiv preprint arXiv:2411.01663*, 2024.

Tri Dao. FlashAttention-2: faster attention with better parallelism and work partitioning. *CoRR*, abs/2307.08691, 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: fast and memory-efficient exact attention with io-awareness. In *NeurIPS*, 2022.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *CoRR*, 2022.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: accurate quantization for generative pre-trained transformers. In *ICLR*, 2023.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling laws for autoregressive generative modeling. *CoRR*, abs/2010.14701, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *ISSCC*, pages 10–14, 2014.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, pages 103–112, 2019.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. *CoRR*, 2023.

Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. Citeseer, 2012.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023.

Zhen Li, Yupeng Su, Runming Yang, Zhongwei Xie, Ngai Wong, and Hongxia Yang. Quantization meets reasoning: Exploring LLM low-bit quantization degradation for mathematical reasoning. *CoRR*, abs/2501.03035, 2025.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. AWQ: activation-aware weight quantization for LLM compression and acceleration. *CoRR*, abs/2306.00978, 2023.

Bin Liu, Fengfu Li, Xiaogang Wang, Bo Zhang, and Junchi Yan. Ternary weight networks. In *ICASSP*, pages 1–5. IEEE, 2023a.

Shih-Yang Liu, Zechun Liu, Xijie Huang, Pingcheng Dong, and Kwang-Ting Cheng. LLM-FP4: 4-bit floating-point quantized transformers. In *EMNLP*, 2023b.

Zechun Liu, Zhiqiang Shen, Shichao Li, Koen Helwegen, Dong Huang, and Kwang-Ting Cheng. How do adam and training strategies help bnns optimization. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 6936–6946. PMLR, 2021.

Zechun Liu, Barlas Oguz, Aasish Pappu, Lin Xiao, Scott Yih, Meng Li, Raghuraman Krishnamoorthi, and Yashar Mehdad. BiT: robustly binarized multi-distilled transformer. In *NeurIPS*, 2022.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. *CoRR*, abs/1809.02789, 2018.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James F. Allen. A corpus and evaluation framework for deeper understanding of commonsense stories. *CoRR*, abs/1604.01696, 2016.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *ACL*. The Association for Computer Linguistics, 2016.

Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, Ruihang Li, Miaosen Zhang, Chen Li, Jia Ning, Ruizhe Wang, Zheng Zhang, Shuguang Liu, Joe Chau, Han Hu, and Peng Cheng. FP8-LM: training FP8 large language models. *CoRR*, abs/2310.18313, 2023.

Haotong Qin, Yifu Ding, Mingyuan Zhang, Qinghua Yan, Aishan Liu, Qingqing Dang, Ziwei Liu, and Xianglong Liu. Bibert: Accurate fully binarized BERT. In *ICLR*. OpenReview.net, 2022.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: imagenet classification using binary convolutional neural networks. In *ECCV*, Lecture Notes in Computer Science, 2016.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: an adversarial winograd schema challenge at scale. In *AAAI*, pages 8732–8740, 2020.

Yuzhang Shang, Zhihang Yuan, Qiang Wu, and Zhen Dong. PB-LLM: partially binarized large language models. *CoRR*, abs/2310.00034, 2023.

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. *CoRR*, abs/2308.13137, 2023.

Noam Shazeer. GLU variants improve transformer. *CoRR*, abs/2002.05202, 2020.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.

Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568: 127063, 2024.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *CoRR*, abs/2307.08621, 2023.

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. *CoRR*, abs/2302.13971, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, and et al. Llama 2: open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023b.

Jonathan Tow, Marco Bellagente, Dakota Mahan, and Carlos Riquelme. Stablelm 3b 4e1t. URL `[https://huggingface.co/stabilityai/stablelm-3b-4e1t](https://huggingface.co/stabilityai/stablelm-3b-4e1t)`.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. *CoRR*, abs/2402.04396, 2024.

Hongyu Wang, Shuming Ma, Shaohan Huang, Li Dong, Wenhui Wang, Zhiliang Peng, Yu Wu, Payal Bajaj, Saksham Singhal, Alon Benhaim, Barun Patra, Zhun Liu, Vishrav Chaudhary, Xia Song, and Furu Wei. Foundation transformers. *CoRR*, 2022.

Lei Wang, Lingxiao Ma, Shijie Cao, Ningxin Zheng, Quanlu Zhang, Jilong Xue, Ziming Miao, Ting Cao, , and Yuqing Yang. Ladder: Efficient tensor compilation on customized data format. In *OSDI*, 2023.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text, NUT@EMNLP 2017*, pages 94–106. Association for Computational Linguistics, 2017.

Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, volume 11217 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2018.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: accurate and efficient post-training quantization for large language models. In *ICML*, 2023.

Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. In *EMNLP-IJCNLP*, 2019.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: can a machine really finish your sentence? In *ACL*, pages 4791–4800, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *NeurIPS*, pages 12360–12371, 2019.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. Ternarybert: Distillation-aware ultra-low bit BERT. In *EMNLP*, pages 509–521. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.EMNLP-MAIN.37.

Yichi Zhang, Zhiru Zhang, and Lukasz Lew. PokeBNN: A binary pursuit of lightweight accuracy. In *CVPR*, pages 12465–12475. IEEE, 2022.

Yichi Zhang, Ankush Garg, Yuan Cao, Lukasz Lew, Behrooz Ghorbani, Zhiru Zhang, and Orhan Firat. Binarized neural machine translation. *CoRR*, 2023.