Rethinking 1-bit Optimization Leveraging Pre-trained Large Language Models

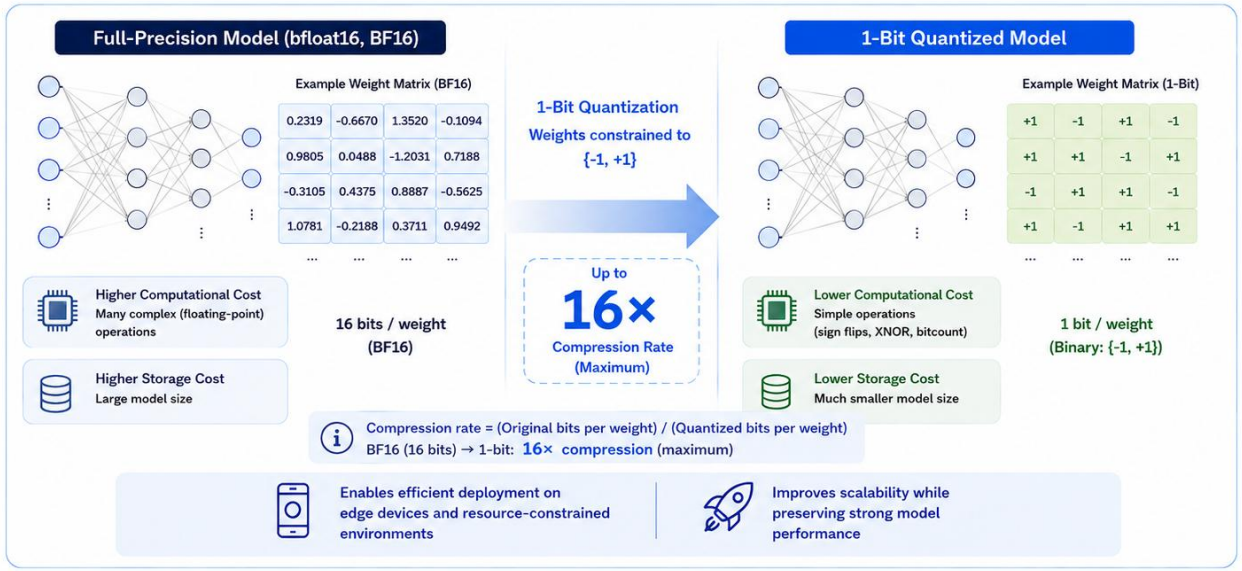
No. 19244

Zhijun Tu, Jian Li, Yuanyuan Xi, Siqi Liu, Chuanjian Liu,
Hanting Chen, Jie Hu, Yunhe Wang

Huawei Technologies Co., Ltd.

Background & Motivation

- The impressive performance of LLMs comes with substantial computational and storage requirements creating challenges for their deployment and scalability, especially in resource-constrained environments.
- 1-bit quantization stands out as an extreme approach, constraining the weights to binary values, typically {-1, +1}, which results in the highest possible compression rate.



Background & Motivation

- PTQ suffers from hardware inefficiency and sever degradation.
- Training from scratch is less efficient.
- Naive quantization is not friendly to pre-training weights.

Models	BitNet (Wang et al., 2023)	OneBit (Xu et al., 2024)	FBI-LLM (Ma et al., 2024a)
#Init	Scratch	Pre-trained	Scratch
#Bit	1.58	1	1
Train Set	RedPajama	Synthetic data	Amber
#Tokens	100 B	13.5 B	1.26 T
#Batch	1 M	128	3.9M
#GPU hours	5.3 k	1.3 k	262 k

Table 1. 1-bit training settings on 7B LLMs.

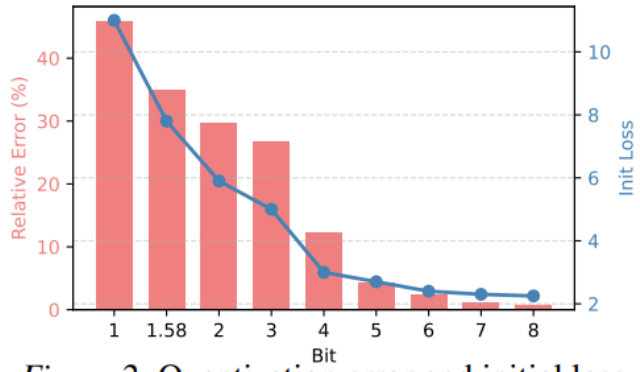


Figure 2. Quantization error and initial loss.

What is the Model Binarization

- **Forward**: The BF16 parameters are mapped to {-1, +1} with an additional scaling factor introduced for error compensation, which is mathematically optimized to minimize the Mean Squared Error (MSE) between the full-precision and 1-bit parameter distributions.

$$X^b = \mathcal{B}(X) = S_a \times \texttt{Sign}(X),$$

where

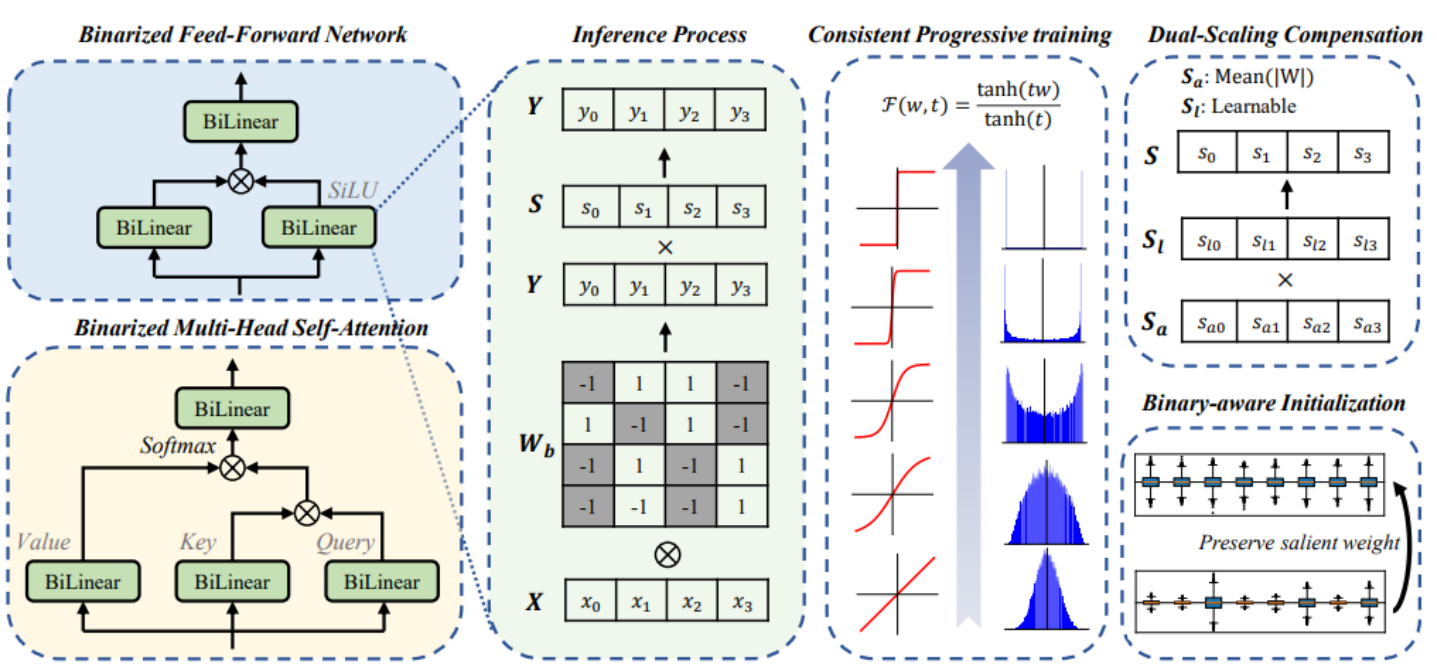
$$\texttt{Sign}(X_i) = \begin{cases} +1, & \text{if } X_i \geq 0, \\ -1, & \text{if } X_i < 0, \end{cases}$$

$$S_a = \frac{1}{n} \sum_{i=1}^{n} |X_i|.$$

- **Backward**: STE circumvents the non-differentiable bottleneck of the sign function by propagating gradients unaltered to update the underlying full-precision weights.

$$g_X = \frac{\partial \mathcal{L}}{\partial X} = \frac{\partial \mathcal{L}}{\partial X^b} \frac{\partial X^b}{\partial X} \approx \frac{\partial \mathcal{L}}{\partial X^b} = g_{X^b},$$

How to Build An Accurate Binarized Large Language Model



- Training based on pretrained parameters instead of training from scratch.
- Protecting the prior knowledge:
  - ✓ Consistent progressive training
  - ✓ Binary-aware Initialization
  - ✓ Dual-Scaling Compensation

Contribution 1: Consistent Progressive Training

Ideally, we aim to design a progressive approximation function

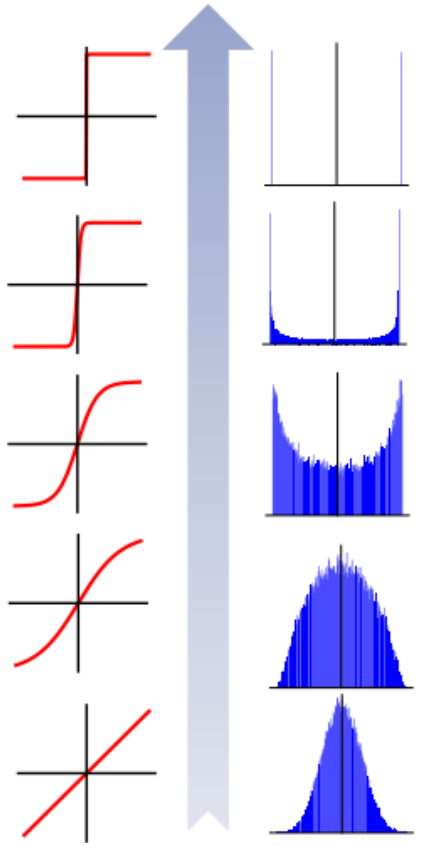
$$\lim_{t\to 0} \mathcal{F}(x,t) \approx x,$$
$$\lim_{t\to \infty} \mathcal{F}(x,t) \approx \mathtt{Sign}(x).$$

Forward:

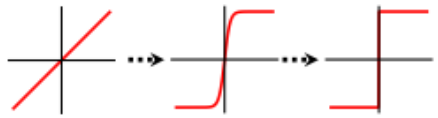
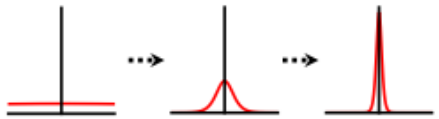
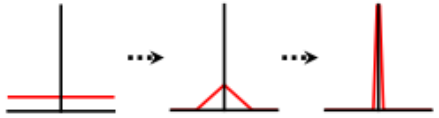
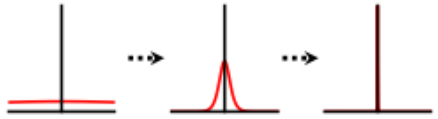
$$\mathcal{F}(x,t) = \mathtt{tanh}(tx)/\mathtt{tanh}(t)$$

Backward:

$$\mathcal{F}'(x,t) = \frac{\partial \mathcal{F}(x,t)}{\partial x} = t(1-\mathtt{tanh}^2(tx))/\mathtt{tanh}(t)$$



Contribution 1: Consistent Progressive Training

IR-Net (Train from Scratch)	**RBNN** (Train from Scratch)	**BinaryLLM** (Train from Pre-trained Models)
Forward		
		
$\mathcal{B}(x) = \text{sign}(x)$	$\mathcal{B}(x) = \text{sign}(x)$	$\mathcal{F}(x,t) = \tanh(tx)/\tanh(t)$
Backward		
		
$g(x) = k \cdot t(1 - \tanh^2(tx))$	$g(x) = \max(k \cdot \sqrt{2}t - \lvert t^2 x \rvert, 0)$	$g(x) = t(1 - \tanh^2(tx))/\tanh(t)$

- Preserves pre-trained knowledge in the initial phase and transitions seamlessly into a 1-bit state in the final stage.
- Enforces mathematical consistency between forward and backward passes to circumvent STE-induced approximation errors

Contribution 2: Binary-aware Initialization

- **GPTQ**: Suffers from inaccurate error compensation in ultra-low bit scenarios and irreversibly distorts the pre-trained weight distribution.
- **AWQ**: Suffers from progressive layer-wise feature error accumulation, which cannot be resolved by its localized, layer-wise optimization search.

$$S_t^* = \arg\min_{S_t} \|\mathcal{B}(W \cdot S_t^{-1})(S_t \cdot A) - WA\|,$$

$$S_t^* = \arg\min_{S_t} \sum_{i}^{L} \log(p(A_i|A_{i-L}, \ldots, A_{i-1}); \mathcal{B}(W \cdot S_t^{-1}))$$

- **Binary-aware Initialization**
  - **Inter-Layer Error Awareness**: Explicitly accounts for progressive error accumulation across different layers during initialization.
  - **Weight Preservation**: Freezes original pre-trained weights to completely avoid irreversible distribution corruption.
  - **Highly Efficient Search**: Optimizes only a small set of scaling factors (S_t), requiring just a few dozen search steps.

Contribution 3: Dual-Scaling Compensation

- **Flaw of Purely Learnable Scales**: While introducing learnable scaling factors (like in FBI-LLM) aims to improve training flexibility, making them purely learnable causes severe accuracy drops and training instability when applied to 1-bit pre-trained LLMs.
- **Indispensable Role of S_a:** The analytical scaling factor S_a is critical because it mathematically guarantees the minimal L_2 quantization error at each individual training step; updating it solely via loss gradients destroys this optimal boundary.
- **Proposed Dual-Scaling Scheme**: To resolve this bottleneck, a dual-scaling compensation scheme is proposed for forward propagation, which retains both the analytical scale S_a, for error minimization) and a new learnable scale S_l, for optimization flexibility) while maintaining inference efficiency.

Training $W^b = S_l \times S_a \times \mathcal{F}(W/S_a, t),$

Inference $W^b = S \times \mathtt{Sign}(W)$ where $S = S_l \times S_a$

Experimental Results

Model	Size	Bit	Perplexity ↓				Zero-shot Accuracy ↑							
			Wiki2	C4	PTB	Avg	BoolQ	PIQA	HS	WG	ARC-e	ARC-c	OBQA	Avg
OPT (Allal et al., 2025)	125M	16.0	27.7	26.6	39.0	31.1	55.4	62.0	31.3	50.3	40.0	22.8	28.0	41.4
SmolLM (Allal et al., 2025)	135M	16.0	17.6	22.1	36.3	25.3	60.2	68.2	42.6	53.0	56.3	28.8	34.4	49.1
Pythia (Biderman et al., 2023)	160M	16.0	26.6	28.9	45.5	33.6	51.3	62.1	31.3	49.6	39.1	24.0	26.8	40.6
FBI-LLM (Ma et al., 2024a)	130M	1.01	28.2	26.9	136.6	63.9	62.1	59.3	28.7	51.0	34.9	20.5	26.4	40.4
BinaryLLM (Ours)	135M	1.01	26.8	28.1	51.1	35.3	60.4	62.0	31.2	51.8	41.5	24.2	28.6	42.8
OPT (Zhang et al., 2022)	1.3B	16.0	14.6	16.1	20.3	17.0	57.8	72.5	53.7	59.5	51.0	29.5	33.4	51.1
LLaMA3 (Dubey et al., 2024)	1.3B	16.0	9.8	14.0	17.6	13.8	64.0	74.5	63.7	60.5	60.4	36.4	26.6	55.2
BitNet (Ma et al., 2024b)	1.3B	1.59	24.1	21.8	145.1	63.7	56.7	68.8	37.7	55.8	54.9	24.2	19.6	45.4
OneBit-OPT (Xu et al., 2024)	1.3B	1.02	25.4	23.0	-	-	59.5	62.6	34.3	51.1	41.3	24.1	-	-
FBI-LLM (Ma et al., 2024a)	1.3B	1.01	12.6	13.8	39.3	21.9	60.3	69.0	42.3	54.0	43.6	25.3	29.6	46.3
BinaryLLM (Ours)	1.3B	1.01	14.7	18.4	27.0	20.0	60.4	70.0	49.6	56.4	47.4	26.0	30.8	48.7
OPT (Zhang et al., 2022)	2.7B	16.0	12.5	14.3	18.0	14.9	60.4	74.8	60.6	61.0	54.4	31.3	35.2	54.0
Pythia (Biderman et al., 2023)	2.8B	16.0	10.2	14.6	18.3	14.4	64.7	73.7	59.3	60.1	58.8	33.0	35.6	55.0
LLaMA3 (Dubey et al., 2024)	3.0B	16.0	7.8	13.5	11.3	10.9	73.4	77.5	73.6	69.9	71.6	46.0	43.0	65.0
OneBit-OPT (Xu et al., 2024)	2.7B	1.02	21.9	20.8	-	-	54.3	63.9	38.2	51.7	43.4	24.4	-	-
BitNet (Ma et al., 2024b)	3.0B	1.59	10.0	9.8	85.0	34.9	61.5	71.5	42.9	59.3	61.4	28.3	61.5	50.2
BinaryLLM (Ours)	3.0B	1.01	12.4	17.1	20.4	16.6	62.2	72.7	58.3	66.4	63.1	34.4	42.0	57.0

Model	Size	Bit	Perplexity ↓			Zero-shot Accuracy ↑							
			Wiki2	C4	Avg	BoolQ	PIQA	HS	WG	ARC-e	ARC-c	OBQA	Avg
LLaMA2 (Touvron et al., 2023)	7B	16	5.5	7.3	6.4	77.7	79.1	76.0	69.1	74.6	46.2	44.2	66.7
OneBit-LLaMA2 (Xu et al., 2024)	7B	1.01	9.7	11.1	10.4	63.1	68.1	52.6	58.4	41.6	29.6	-	-
MoS (Jo et al., 2024)	7B	1.01	7.9	9.8	8.8	65.0	71.6	59.4	56.2	41.8	30.0	-	-
FBI-LLM (Ma et al., 2024a)	7B	1.01	9.1	10.5	10.3	61.5	72.6	57.7	58.9	53.0	29.9	36.8	52.9
BinaryLLM (Ours)	7B	1.01	8.7	9.7	9.2	65.5	73.2	59.5	66.4	60.3	37.2	34.5	56.7

Ablations

Comparison results with different baselines.

Model	Method	Perplexity ↓				Zero-shot Accuracy ↑							
		Wiki2	C4	PTB	Avg	BoolQ	PIQA	HS	WG	ARC-e	ARC-c	OBQA	Avg
Pythia-160M (Biderman et al., 2023)	FP16	26.6	28.9	45.5	33.6	51.3	62.1	31.3	49.6	39.1	24.0	26.8	40.6
	BinaryLLM	35.4	33.2	53.1	40.6	58.5	60.3	29.4	52.3	37.8	22.6	25.2	40.9
	Δ	+8.8	+4.3	+7.6	+7.0	+7.2	-1.8	-1.9	+2.7	-1.3	-1.4	-1.6	+0.3
SmolLM 135M (Allal et al., 2025)	FP16	17.6	22.1	36.3	25.3	60.2	68.2	42.6	53.0	56.3	28.8	34.4	49.1
	BinaryLLM	25.9	27.0	47.6	33.5	61.6	60.9	30.7	50.4	40.9	23.1	28.4	42.3
	Δ	+8.3	+4.9	+11.3	+8.2	+1.4	-7.3	-11.9	-2.6	-15.4	-5.7	-4.0	-6.7

Comparison results with different progressive methods

Method	Perplexity ↓				Zero-shot Accuracy ↑							
	Wiki2	C4	PTB	Avg	BoolQ	PIQA	HS	WG	ARC-e	ARC-c	OBQA	Avg
FP16	17.6	22.1	36.3	25.3	60.2	68.2	42.6	53.0	56.3	28.8	34.4	49.1
vanilla 1-bit	30.4	31.1	52.9	38.1	58.3	59.0	29.3	51.5	37.2	22.8	28.6	41.0
IR-Net (Qin et al., 2020)	29.8	30.7	51.1	37.3	59.2	59.5	30.3	50.3	37.3	23.1	28.3	41.1
BinaryLLM	27.4	28.2	49.3	35.0	61.0	60.6	30.8	50.5	40.3	22.9	28.1	42.0

Ablations

Perplexity of different scaling factors

Methods	Perplexity ↓			
	Wiki2	C4	PTB	Average
S_a	73.1	54.8	98.7	75.5
S_l	76.2	58.5	96.4	77.0
$S_l \times S_a$	68.7	51.5	92.5	70.9

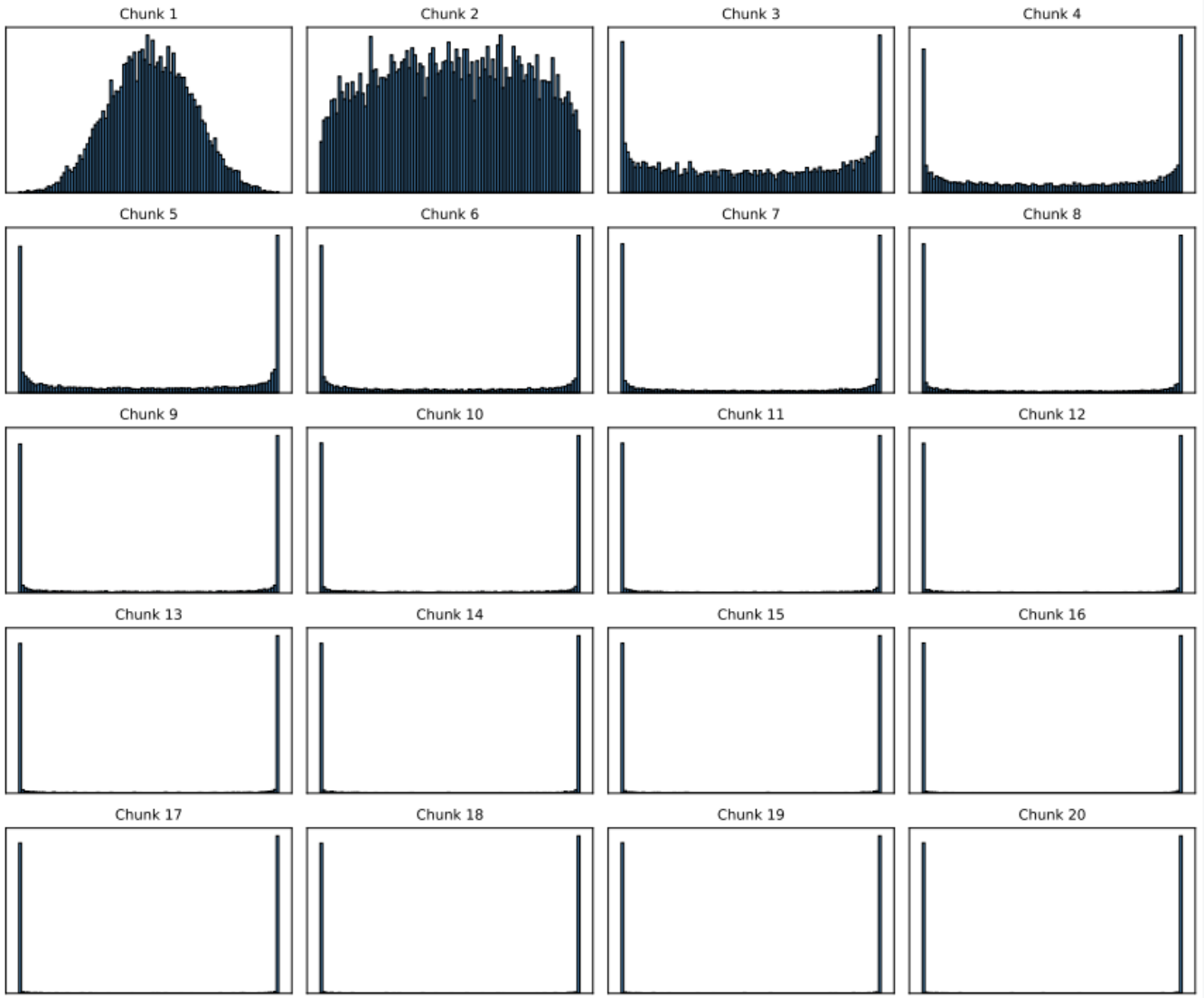
Comparison results with 1-bit PTQ methods

Models	Methods	Bit	Perplexity ↓		
			Wiki2	C4	Average
LLaMA3-1B (Dubey et al., 2024)	Full-Precision	16	17.6	22.1	19.9
	BiLLM (Huang et al., 2024)	1.08	815.4	610.2	712.8
	ARB-LLM-X (Li et al., 2024)	1.08	115.4	144.0	129.7
	BinaryLLM	1.01	**14.7**	**18.4**	**16.6**
LLaMA3-3B (Dubey et al., 2024)	Full-Precision	16	7.8	13.5	10.7
	BiLLM (Huang et al., 2024)	1.08	104.3	82.8	93.6
	ARB-LLM-X (Li et al., 2024)	1.08	49.2	56.6	52.9
	BinaryLLM	1.01	**12.4**	**17.1**	**14.8**

Effectiveness on Binary-aware Initialization

Models	Wiki2	C4	PTB	Avg. PPL
FP Model	9.8	14.0	17.6	13.8
w/ BaI	14.7	18.4	27.0	20.0
w/o BaI	15.1	18.7	27.5	20.4

Visualization of Weights



Efficiency Analysis

Models	Average Bit	Memory (GB)	Cycles (T)
LLaMA2-7B	16.0	13.48	1.72
LLaMA2-7B-INT4	4.0	3.76	0.48
BitNet	1.58	1.80	0.23
BiLLM	1.08	1.40	0.18
OneBit	1.01	1.34	0.17
FBI-LLM	1.01	1.34	0.17
BinaryLLM	1.01	1.34	0.17

Table 8. Memory and cycles of different methods.

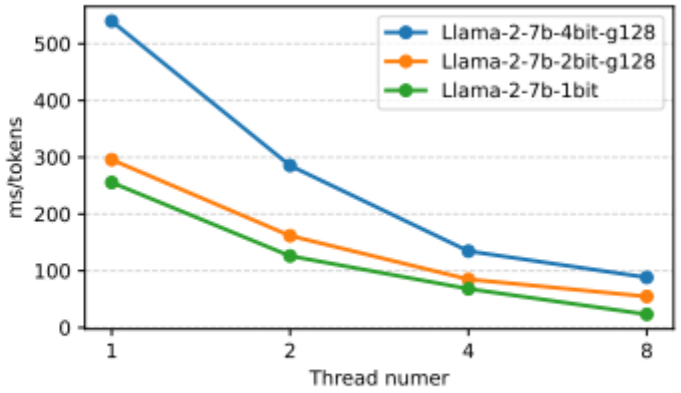


Figure 4. Latency with different threads.

- Inference Framework: T-MAC, a specialized CPU-based inference engine optimized for low-bit LLM edge deployment.
- Hardware Environment: AMD EPYC 7642 48-Core Processor.

Thank you!