ShiftAddLLM: Accelerating Pretrained LLMs via Post-Training Multiplication-Less Reparameterization

Haoran You[†], **Yipin Guo**[†], **Yichao Fu**[†], **Wei Zhou**[†], **Huihong Shi**[†], **Xiaofan Zhang**[*]
Souvik Kundu[◇], **Amir Yazdanbakhsh**[‡], **Yingyan (Celine) Lin**[†]

[†]Georgia Institute of Technology   [◇]Intel Labs   [*]Google   [‡]Google DeepMind
[†]*{haoran.you, celine.lin}@gatech.edu, eic-lab@groups.gatech.edu*
[◇]*souvikk.kundu@intel.com, *[‡]{xiaofanz, ayazdan}@google.com*

Abstract

Large language models (LLMs) have shown impressive performance on language tasks but face challenges when deployed on resource-constrained devices due to their extensive parameters and reliance on dense multiplications, resulting in high memory demands and latency bottlenecks. Shift-and-add reparameterization offers a promising solution by replacing costly multiplications with hardware-friendly primitives in both the attention and multi-layer perceptron (MLP) layers of an LLM. However, current reparameterization techniques require training from scratch or full parameter fine-tuning to restore accuracy, which is resource-intensive for LLMs. To address this, we propose accelerating pretrained LLMs through post-training shift-and-add reparameterization, creating efficient multiplication-free models, dubbed ShiftAddLLM. Specifically, we quantize each weight matrix into binary matrices paired with group-wise scaling factors. The associated multiplications are reparameterized into (1) shifts between activations and scaling factors and (2) queries and adds according to the binary matrices. To reduce accuracy loss, we present a multi-objective optimization method to minimize both weight and output activation reparameterization errors. Additionally, based on varying sensitivity across layers to reparameterization, we develop an automated bit allocation strategy to further reduce memory usage and latency. Experiments on five LLM families and eight tasks consistently validate the effectiveness of ShiftAddLLM, achieving average perplexity improvements of 5.6 and 22.7 points at comparable or lower latency compared to the most competitive quantized LLMs at 3 and 2 bits, respectively, and more than 80% memory and energy reductions over the original LLMs. Codes and models are available at `https://github.com/GATECH-EIC/ShiftAddLLM`.

1 Introduction

Pretrained LLMs have demonstrated state-of-the-art performance in language understanding and generation tasks [42, 43, 53, 3, 67, 51, 52, 2]. However, deploying these LLMs incurs significant hardware demands, including high latency, memory, and energy consumption, especially on edge or cloud GPU devices. The primary bottlenecks are their immense parameter sizes and the associated multiplication operations. For instance, GPT-3, with 175 billion parameters, requires 350GB of memory in FP16 format [34] and performs 10^{15} floating-point operations (FLOPs) for a single forward pass [18]. Previous efforts to improve LLM efficiency have focused on pruning [36, 49, 19, 23, 40], quantization [57, 34, 17, 44], and attention optimization [12, 65, 61]. However, these methods still rely on costly multiplication operations in both the attention and MLP layers.

Preprint. Under review.

Table 1: Hardware cost under 45nm CMOS [25, 63, 22, 46, 5].

OPs	Multiplication				Add				Shift			LUTs
	FP32	FP16	INT32	INT8	FP32	FP16	INT32	INT8	INT32	INT16	INT8	(8-bit Query)
Energy (pJ)	3.7	0.9	3.1	0.2	1.1	0.4	0.1	0.03	0.13	0.057	0.024	0.37 (8 OPs)
Area (μm^2)	7700	1640	3495	282	4184	1360	137	36	157	73	34	787 (8 OPs)

* Note that 1 LUT corresponds to 8 operations, as each bit in queries is from a weight element.

We identify a promising yet unexplored opportunity for improving LLM efficiency: *reparameterizing their extensive multiplications with more cost-effective hardware substitutes, such as bitwise shifts and adds.* Inspired by practices in computer architecture and digital signal processing, replacing multiplications with bitwise shifts and adds [60, 21] can offer up to $31\times$ energy and $26\times$ area reductions (see Tab. 1). This hardware-inspired approach can lead to efficient and fast implementations, as shown by previous research on ShiftAddNet [63, 64, 66]. Unlike previous reparameterization techniques that require training from scratch or extensive fine-tuning, we propose a new method to integrate the shift-and-add concept into LLMs through post-training optimization.

To design multiplication-less LLMs, we need to address three key challenges: ***First***, how can we effectively reparameterize pretrained LLMs with shifts and adds in a post-training manner? Previous reparameterization techniques [63, 66] often result in significant quantization errors, requiring extensive fine-tuning or retraining to avoid accuracy drops. We aim to develop a ready-to-use *post-training* reparameterization method for LLMs. ***Second***, how can we mitigate the accuracy drop from shift-and-add reparameterization? Approximating original multiplications with lower-bit shifts and adds typically reduces accuracy. Most studies resort to fine-tuning or increasing model sizes, complicating LLM deployment. We hypothesize that optimizing both weight and activation errors can minimize overall reparameterization error, aligning with recent activation-aware weight quantization methods in LLMs. ***Third***, how can we handle varying sensitivities to reparameterization across different layers and blocks in LLMs? An automated strategy to determine the optimal number of bits for reparameterized weights in each layer is needed. More vulnerable layers should have higher-bit representations, while less sensitive layers can use lower-bit representations. This ensures no bottlenecked layers due to aggressive reparameterization and maximizes redundancy exploitation. To the best of our knowledge, *this is the first attempt* to address these three challenges for multiplication-less LLMs through *post-training* reparameterization. Our contributions are summarized as follows:

- We propose accelerating pretrained LLMs via a *post-training* bitwise shift-and-add reparameterization, resulting in efficient multiplication-less LLMs, dubbed ShiftAddLLM. All weights are quantized into binary matrices paired with group-wise scaling factors; the associated multiplications are reparameterized into shift-and-add operations.

- To mitigate accuracy loss, we present a multi-objective optimization method aligning and optimizing both weight and output activation objectives, minimizing overall reparameterization error, and achieving lower perplexity and better task accuracy.

- We introduce a mixed and automated bit allocation strategy that determines the optimal number of bits for reparameterized weights per layer, based on their vulnerability to compression. Susceptible layers receive higher-bit representations, while less sensitive ones get lower-bit representations.

Our extensive results across five LLMs and eight tasks consistently show the superior accuracy and efficiency trade-offs achieved by ShiftAddLLM, with average perplexity improvements of 5.6 and 22.7 at comparable or even lower latency compared to the most competitive quantized LLMs at three and two bits, respectively, and more than 80% memory and energy reductions over the original LLMs.

2 Related Works

LLM Quantization. Significant efforts have been made to quantize LLMs, including quantization-aware training (QAT) [35, 47] and post-training quantization (PTQ) [17, 34, 57, 14]. QAT requires calibrated data and significant retraining resources, whereas PTQ is more dominant due to it lower computational and time overhead. There are two prevalent PTQ strategies for LLMs: *(1)* uniform quantization of both weights and activations [57, 14, 62], often limited to 8 bits (W8A8) as lower bit representations can significantly reduce accuracy; and *(2)* lower bit weight-only quantization [17, 44, 13, 26, 6], which quantizes LLM weights to lower bits while keeping activations in a FP16 format.

2

This approach alleviates memory bottlenecks associated with the vast parameters of LLMs. For instance, GPTQ [17] uses gradient-based weight quantization and develops `INT3/4` kernels to reduce data movements, and LUT-GEMM [44] eliminates the dequantization and uses custom LUT-based CUDA kernels to reduce memory and computation costs. In contrast, ShiftAddLLM is the first to employ the shift-and-add idea for reparameterizing pre-trained LLMs. This reparameterization reduces bit usage for weights and replaces costly multiplications with hardware-friendly primitives, further reducing energy, latency, and memory.

Multiplication-less Models. The efficient model community has focused on reducing or replacing multiplications. In CNNs, binary networks [10, 30] binarize weights and activations, while shift-based networks use spatial shifts [56] or bitwise shifts [15] to substitute for multiplications. AdderNet [7, 59, 55] replaces multiplications with additions, albeit with a small accuracy drop. ShiftAddNet [63] reparameterizes CNNs with cascaded shift and add layers. These techniques have been adapted to Transformers. BiLLM [26] introduces binary LLMs, while [48] and [54] extend the addition or shift concepts to the attention mechanisms, respectively. ShiftAddViT [66] reparameterizes pretrained Vision Transformers (ViTs) with shifts and adds. Contemporary work MatMul-free LM [69] leverages additive operators and Hadamard products for multiplication-free language model training, relying on FPGAs for speedups. Compared to closely related works like ShiftAddNet [63] and MatMul-free LM [69], which requires training from scratch, and ShiftAddViT [66], which demands extensive parameter fine-tuning, ShiftAddLLM applies the shift-and-add concept to pre-trained LLMs without additional training or fine-tuning. We also use a multi-objective optimization and automated bit allocation strategy to further improve accuracy or reduce GPU latency, energy, and memory usage.

3 Preliminaries

Binary-coding Quantization (BCQ). BCQ [58] quantizes each weight tensor in an L-layer LLM $\mathbf{w} \in \mathbb{R}^{m \times n}$ into q bits using a linear combination of binary matrices $\{\mathbf{b}_i\}_{i=1}^q$ and corresponding scaling factors $\{\alpha_i\}_{i=1}^q$, where $\mathbf{b}_i \in \{-1, 1\}^{m \times n}$. The weights are then approximated by $\mathbf{w}_q = \sum_{i=1}^q \alpha_i \mathbf{b}_i$ as a result of minimizing the quantization error, i.e., $\arg\min_{\alpha_i, \mathbf{b}_i} \|\mathbf{w} - \sum_{i=1}^q \alpha_i \mathbf{b}_i\|^2$ to obtain the optimal $\alpha_i^*, \mathbf{b}_i^*$. If q is 1, then the problem collapses to binary quantization, which has an analytical solution: $\mathbf{b}^* = \text{sign}(\mathbf{w}), \alpha^* = \mathbf{w}^\top \mathbf{b}^*/n$. For multi-bit quantization, we resort to greedy and alternating methods [58, 28, 31], as shown in Alg. 1. Initially, we use the greedy method [20] to initialize $\alpha_i, \mathbf{b}_i$, where the i-th bit quantization is performed by minimizing the residual $\mathbf{r}$ from the $(i-1)$-th bit:

Algorithm 1 Alternating Multi-bit BCQ [58]

1: **Input:** Full-precision weight $\mathbf{w} \in \mathbb{R}^n$, bit-width q, alternating cycles T
2: **Output:** $\alpha_i^*, \mathbf{b}_i^* \in \{-1, 1\}^{m \times n}$
3: **Function** MULTI-BIT BCQ$(\mathbf{w}, q, T)$
4:     $\{\alpha_i, \mathbf{b}_i\}_{i=1}^q \leftarrow$ GREEDY$(\mathbf{w})$
5:     **for** $t \leftarrow 1$ to T **do**
6:         $\{\alpha_i\}_{i=1}^q \leftarrow$ LS$(\mathbf{B}, \mathbf{w})$
7:         $\{\mathbf{b}_i\}_{i=1}^q \leftarrow$ BS$(\alpha_1, \ldots, \alpha_q, \mathbf{w})$
8:     **end for**
9: **end Function**

$$\min_{\alpha_i, \mathbf{b}_i} \|\mathbf{r}_{i-1} - \alpha_i \mathbf{b}_i\|^2, \quad \text{where} \quad \mathbf{r}_{i-1} = \mathbf{w} - \sum_{j=1}^{i-1} \alpha_j \mathbf{b}_j, \quad 1 < i \leq q. \tag{1}$$

We then obtain the initialized $\alpha_i, \mathbf{b}_i$ sequentially as $\mathbf{b}_i = \text{sign}(\mathbf{r}_i)$ and $\alpha_i = \mathbf{r}_i^\top \mathbf{b}_i/n$ (**Line 4**). Next, we perform alternating optimization to further minimize the quantization error. Specifically, $\{\alpha_i\}_{i=1}^q$ can be iteratively refined using ordinary least squares (LS) [20] as $[\alpha_1, ..., \alpha_q] = ((\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{w})^\top$, where $\mathbf{B} = [\mathbf{b}_1, ..., \mathbf{b}_q] \in \{-1, 1\}^{m \times n \times q}$ (**Line 6**). The binary codes $\{\mathbf{b}_i\}_{i=1}^q$ can then be iteratively recalibrated using a binary search (BS) given the refined $\{\alpha_i\}_{i=1}^q$ (**Line 7**) [58].

Such BCQ can support both uniform and non-uniform quantization formats by adjusting the scaling factors and biases accordingly [44]. Our ShiftAddLLM is built on top of BCQ but further replaces all associated multiplications with lower-cost hardware substitutes (e.g., shifts, adds, and LUT queries). We optimize not only the weight quantization error but also the output activation error, thereby achieving lower quantization bits along with savings in energy, memory, and computational costs.

Shift and Add Primitives. Direct hardware implementation of multiplications is often inefficient. Using shift and add operations as "shortcuts" provides a more efficient alternative. Shifts, which are
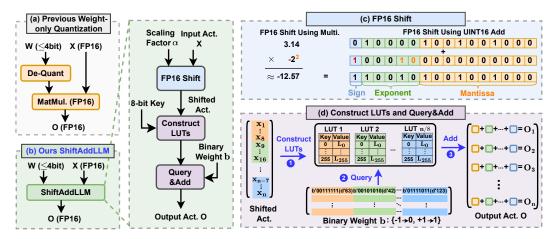
Figure 1: Illustration of our proposed post-training reparameterization for ShiftAddLLM.

equivalent to multiplying by powers of two, offer a non-uniform quantization solution and can result in significant savings. For example, reductions of $23.8\times$ and $22.3\times$ in energy and area, respectively, as shown in Tab. 1. Similarly, additions can achieve up to $31.0\times$ and $25.5\times$ savings in energy and area. Both primitives have inspired many innovations in efficient model innovations [7, 15, 63, 66].

4   The Proposed ShiftAddLLM Framework

Overview. We introduce our ShiftAddLLM as follows: *First*, we describe the reparameterization of pretrained LLMs through a *post-training* shift-and-add approach in Sec. 4.1. *Second*, to enhance accuracy, we introduce a multi-objective optimization method that accounts for both weight quantization error and output activation error, detailed in Sec. 4.2. *Third*, to improve efficiency, we explore a mixed and automated bit allocation strategy, illustrated in Sec. 4.3.

4.1   ShiftAddLLM: Post-training Reparameterization of LLMs with Shift and Add Primitives

Post-training Reparameterization of LLMs. To avoid the need for fine-tuning after reparameterization, our method closely mimics the original multiplications used in LLMs. Previous methods, such as weight-only quantization techniques [17], employ gradient-based or activation-aware uniform quantization to fit the pretrained weight distribution better, thereby achieving lower quantization errors. However, these methods often lack direct hardware support and require on-the-fly dequantization to FP16 for multiplication with activations, as depicted in Fig. 1 (a). In contrast, our ShiftAddLLM uses the BCQ format, supporting non-uniform quantization with customized CUDA kernels [44, 27], bypassing the need for dequantization, as illustrated in Fig. 1 (b). In particular, our method employs the Alg. 1 to quantize pretrained weights into binary matrices $\{\mathbf{b}_i\}_{i=1}^{q}$ and scaling factors $\{\alpha_i\}_{i=1}^{q}$. Note that during the alternating optimization cycles, we further quantize all scaling factors to powers of two (PoT) [33], as described by the equation:

$$\alpha_k = \text{POT}\left(\mathbf{r}_{k-1}\right) = \text{POT}(\alpha - \sum_{j=0}^{k-1} \alpha_j), \quad \text{where} \quad \text{POT}(\alpha) = \text{sign}(\alpha) \cdot 2^{\mathbf{P}}, \quad 1 \le k \le K. \quad (2)$$

This additive PoT method adopts a greedy strategy to enhance the representational capacity of PoT, using K scaling factors, where the k-th PoT minimizes the residual $\mathbf{r}$ of the $(k-1)$-th PoT. Each PoT effectively quantizes the scaling factor α into $\text{sign}(\alpha) \cdot 2^{\mathbf{P}}$, where $\text{sign}(\alpha)$ indicates sign flips, $\mathbf{P} = \text{round}(\log_2(\text{abs}(\alpha)))$, and $2^{\mathbf{P}}$ denotes a bitwise shift to the left ($\mathbf{P} > 0$) or right ($\mathbf{P} < 0$).

After the above reparameterization, we can then replace the associated multiplication between weights and activations into two steps: (1) Bitwise shifts between activations and scaling factors. Note that the activation is still in the FP16 format, and the multiplication between a floating-point number and a positive or negative PoT integer can be efficiently implemented by an integer addition instruction on existing hardware following DenseShift [32], as also illustrated in Fig. 1 (c); (2) Queries and adds
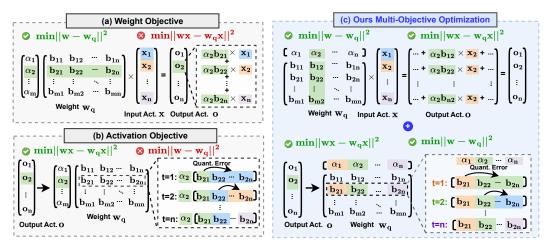
Figure 2: Illustration of our proposed multi-objective optimization framework.

intermediate shifted activations with the binary matrices. To implement this efficiently and reduce redundant additions or accumulations, as shown in Fig. 1 (d), we pre-compute 256 ($= 2^8$) possible values for every eight elements in the shifted activations to construct LUTs. This allows us to handle the multiplication between the binary matrix $\mathbf{b}_i$ and the shifted activations as queries to the LUTs. We then add all the partial sums to obtain the final output activations. Such LUTs are well supported by existing GPU kernels [44, 27]. The reparameterization can be applied to all weights in pretrained LLMs in a *post-training* manner, replacing costly multiplications with efficient hardware operations.

Insights. ShiftAddLLM presents a novel *multiplication-less* approach that leverages non-uniform quantization via BCQ and additive PoT. This methodology enhances the representation capacity for outlier weights and activations of large magnitude compared to uniform quantization. Moreover, additive PoT effectively resolves the issue of limited quantization resolution for non-outlier weights and activations. Overall, it allows the quantization levels to better align with the data distribution.

4.2 ShiftAddLLM: Multi-objective Optimization

Motivating Analysis on Previous LLM Quantization Objectives. We examine previous weight-only quantization methods to understand the causes of large quantization error and accuracy drop. These methods typically use either a *weight* or *activation* objective to minimize quantization error. Specifically, the "*weight objective*" (see Fig. 2 (a)) aims to minimize the weight quantization error, i.e., $\|\mathbf{W} - \mathbf{W}_q\|^2$, and adopts scaling factors for each row of quantized weights. However, this does not optimize output activation error, as each weight element is multiplied by a unique input activation before summing to produce the output. Varying input activations, especially outliers [57, 34], rescale the weight quantization error differently, causing significant divergence in the output activation. For example, LUT-GEMM [44] adopts this weight objective. On the other hand, the "*activation objective*" (see Fig. 2 (b)) minimizes the output activation error, i.e., $\|\mathbf{W}\mathbf{X} - \mathbf{W}_q\mathbf{X}\|$, by quantizing one column of weights at a time and continuously updating the remaining unquantized weights to compensate for the quantization error incurred by quantizing a single weight column. However, the fixed scaling factors may not adequately accommodate the weights adjusted afterward. OPTQ [17] employs this activation objective.

Our Multi-Objective Optimization. To further mitigate accuracy drop after reparameterization (see Sec. 4.1), we introduce a multi-objective optimization framework that combines weight and activation objectives using column-wise scaling factors. This framework effectively reduces quantization error for both weights and activations, thereby improving the accuracy of ShiftAddLLM.

As shown in Fig. 2 (c), using column-wise scaling factors overcomes the limitations of the previous weight objective [44] by eliminating the impact of varying input activations on quantized weights. Each scaling factor corresponds to a constant activation value. Additionally, scaling factors for subsequent columns are updated gradually after compensating for the corresponding column's weights, ensuring a better fit than the previous activation objective [17].

5

Accuracy vs. Latency Tradeoffs. The column-wise scaling factor design significantly boosts accuracy after reparameterization. However, it does not fully leverage BCQ [44, 27], which process eight elements per row of weights in parallel as LUT keys, resulting in latency overhead for models with $\geq$30B param-

(a) Block-wise Scaling Factors

(b) Design Comparisons

OPT-30B w/	PPL ($\downarrow$)	Lat. (ms)
Row-wise	16.3 (✗)	33.2 (✓)
Column-wise	9.6 (✓)	44.1 (✗)
Block-wise	9.9 (✓)	33.5 (✓)

Figure 3: (a) the block-wise scaling factors and (b) the comparison among different designs on OPT-30B [67].

eters. For example, testing on the OPT-30B [67] model and WikiText-2 dataset [37] showed 6.7 perplexity reduction but with a 24.7% latency overhead, as shown in Fig. 3 (b).

To address this, we propose a block-wise scaling factor design that groups 8 columns and ⅛ of the original rows to share a scaling factor, ensuring compatibility with the BCQ kernel and achieving latency reductions, as shown in Fig. 3 (a). We refer to ShiftAddLLM with column-wise scaling factors as *"Ours (Acc.)"* for high accuracy optimization, and with block-wise scaling factors as *"Ours (Lat.)"* for optimized accuracy-latency trade-off.

Insights. Our multi-objective optimization approach integrates both weight and activation objectives, reducing weight quantization error in an activation-aware manner and output activation error reduction in a weight-aware manner. This synergy, achieved through a simple column-wise or block-wise design, significantly boosts the accuracy of weight-only quantization. This aligns with the principles of previous activation-aware weight quantization methods [34].

4.3 ShiftAddLLM: Mixed and Automated Bit Allocation

Sensitivity Analysis. We analyze the sensitivity of different layers and blocks in LLMs to shift-and-add reparameterization. As shown in Fig. 4, later blocks incur more quantization or reparameterization errors. Within each block, Q/K linear layers are generally more sensitive to reparameterization than other linear layers. This diverse sensitivity motivates us to explore mixed bit allocations for LLM reparameterization and develop strategies to automatically determine the optimal bit allocations given the average bit budgets.
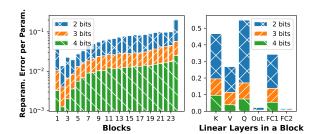

Figure 4: Sensitivity analysis on OPT-1.3B [67].

Criteria and Automated Bit Allocation. To develop the bit allocation scheme, we propose criteria to estimate the importance of linear weights and formulate the bit allocation as an integer programming problem. For weight $\mathbf{W}_i$ from the i-th layer of an LLM, the criterion C_i is defined as follows:

$$C_i = \|\text{IS}\|_F \cdot \text{STD}(\text{IS})^2, \quad \text{where}$$
$$\text{IS} = \mathbf{W}_i/\text{diag}(\text{cholesky}((\mathbf{X}_i\mathbf{X}_i^T)^{-1})), \quad (3)$$

where the importance score (IS) is inspired by Optimal Brain Compression [24, 16, 17] and is correlated to the increase in the quadratic reconstruction error $\|\mathbf{WX} - \mathbf{W}_q\mathbf{X}\|^2$ after reparameterizing the weights, i.e., IS $\uparrow$, error increases $\downarrow$. The F-norm of IS indicates the overall importance of $\mathbf{W}_i$, while the standard deviation (STD) highlights the reparameterization difficulty for outliers. Considering both factors, we achieve a more effective evaluation metric proportional to the actual reparameterization
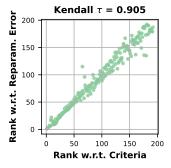

Figure 5: Rank comparisons.

error. As shown in Fig. 5, the rankings derived from our defined criteria and the actual reparameterization error are highly correlated, with a Kendall τ of 0.905. To refine the criteria by incorporating the bit-width, we use least squares polynomial fits to estimate each bit's corresponding reparameterization error as $C_{i,b}$.

Table 2: Perplexity comparisons of the OPT models on WikiText-2. Note that we set the group size of all methods as the length of rows following the setting of OPTQ [17] for a fair comparison.

OPT (PPL ↓)	Bits	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B
FP16	16	27.65	22.00	14.62	12.47	10.86	10.13	9.56	9.34
OPTQ [17]	3	53.85	33.79	20.97	16.88	14.86	11.61	10.27	14.16
LUT-GEMM [44]	3	60.00	42.32	49.10	17.55	17.44	12.50	139.90	100.33
AWQ [34]	3	54.75	35416.00	24.60	39.01	16.47	16.53	31.01	5622.00
Ours (Acc.)	3	**31.29**	**24.24**	**21.53**	**13.68**	**11.18**	**10.39**	**9.63**	**9.43**
OPTQ [17]	2	2467.50	10433.30	4737.05	6294.68	442.63	126.09	71.70	20.91
LUT-GEMM [44]	2	4844.32	2042.90	3851.50	616.30	17455.52	4963.27	7727.27	6246.00
AWQ [34]	2	3514.61	18313.24	9472.81	22857.70	8168.30	5014.92	7780.96	103843.84
QuIP [6]	2	92.84	146.15	27.90	30.02	16.30	12.34	11.48	10.92
Ours (Acc.)	2	**51.15**	**40.24**	**29.03**	**20.78**	**13.78**	**12.17**	**10.67**	**10.33**

Given the criteria, we can formulate the automated bit allocation as an integer programming problem:

$$\arg\min_{\beta_{i,b}} \sum_i^L \sum_b \beta_{i,b} \cdot C_{i,b}, \quad \text{s.t.} \quad \sum_b \beta_{i,b} = 1, \quad \sum_i^L \sum_b \beta_{i,b} \cdot b \leq \mathcal{B} \cdot L, \quad (4)$$

where L is the number of layers in the target LLM, $b \in \{2, 3, 4\}$ denotes the available bit widths, and $\beta_{i,b} \in \{0, 1\}$ is the one-hot indicator for the i-th layer to determine the assigned bits, e.g., $\{0, 1, 0\}$ means 3 bits. The objective is to minimize the summed criteria C of all layers under the given average bit budget $\mathcal{B}$ per layer. The final $\beta_{i,b}$ represents the assigned bits for the i-th layer in the target LLM.

Insights. Using mixed bits instead of static ones can improve the accuracy-efficiency tradeoffs by adapting the varying sensitivities across layers, e.g., Q/K linear layers exhibit higher sensitivity to reparameterization; Our adopted criteria provide a quick estimation of the reparameterization error.

5 Experiments

5.1 Experiment Settings

Models. We consider five representative SOTA LLM families, including OPT [67], LLaMA-1/2/3 [52, 2], Gemma [38], Mistral [29], and Bloom [45]. ***Tasks and Datasets.*** We evaluate all five LLMs on the commonly adopted language modeling task using the WikiText-2 [37] dataset for perplexity measurement. Additionally, we extend the evaluation of the two largest models, OPT-66B and LLaMA-2-70B, to seven downstream tasks for zero-shot task accuracy evaluation. These tasks include ARC (Challenge/Easy) [4], BoolQ [9], Copa [1], PIQA [50], RTE [11], and StoryCloze [39]. ***Baselines.*** We consider four SOTA LLM quantization methods: OPTQ [17], LUT-GEMM [44], QuIP [6], and AWQ [34]. ***Evaluation Metrics.*** We evaluate ShiftAddLLM and the baselines using both accuracy and efficiency metrics. For accuracy, we evaluate perplexity on the WikiText-2 dataset and zero-shot accuracy on seven downstream tasks. For efficiency, we measure the latency on a single A100-80GB GPU (PCIe) [41] and estimate the energy costs using an Eyeriss-like hardware accelerator [8, 68], which calculates not only computational but also data movement energy.

5.2 ShiftAddLLM over SOTA LLM Quantization Baselines

Results on OPT Models. To evaluate the effectiveness of our ShiftAddLLM, we compare with four SOTA LLM quantization baselines: OPTQ [17], LUT-GEMM [44], QuIP [6], and AWQ [34]. Using the OPT model family [67] and the WikiText-2 dataset [37], we assess perplexity, GPU latency, and energy costs. As shown in Tab. 2, *Ours (Acc.)* consistently outperforms all baselines, achieving an average perplexity reduction of 5.63/38.47/5136.13 compared to OPTQ, LUT-GEMM, and AWQ, respectively, at 3 bits. At 2 bits, where most baselines fail with significantly high perplexity, our method maintains low perplexity, and achieves an average 22.74 perplexity reduction over the most competitive QuIP. Also, as shown in Fig. 6 (a & b), *Ours (Lat.)* consistently achieves better accuracy-latency tradeoffs, achieving 0.91∼103830.45 perplexity reduction at comparable latency or 6.5%∼60.1% latency reductions and 26.0%∼44.7% energy savings at similar or even lower perplexity. Complete quantitative data on accuracy, latency, and energy are provided in Appendix A.
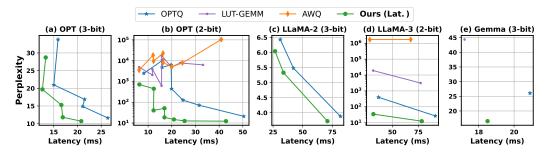
Figure 6: Accuracy-latency tradeoff comparisons on the OPT, LLaMA-2/3, and Gemma models.

Table 5: Accuracy comparisons on seven downstream tasks for OPT-66B and LLaMA-2-70B.

Models	Methods	Bits	ARC_C	ARC_E	Copa	BoolQ	PIQA	Storycloze	RTE	Mean
	OPTQ [17]	3	24.66	48.86	70	52.05	64.47	67.09	53.07	51.99
OPT-66B	LUT-GEMM [44]	3	24.15	51.85	81	53.52	61.97	60.60	48.74	51.97
	Ours (Acc.)	3	**35.24**	**70.88**	**87**	**72.45**	**77.64**	**77.15**	**63.18**	**65.58**
	OPTQ [17]	3	45.82	76.34	90	81.74	79.71	77.34	67.51	74.07
LLaMA-2-70B	LUT-GEMM [44]	3	47.70	76.42	89	80.31	80.20	77.78	68.59	74.29
	Ours (Acc.)	3	**48.38**	**77.06**	**93**	**84.25**	**80.47**	**78.49**	**75.09**	**76.68**

Results on LLaMA Models. We further evaluate ShiftAddLLM on LLaMA models [51, 52, 2] due to their superior performance among open-source LLMs. As shown in Tab. 3, *Ours (Acc.)* consistently outperforms all baselines, achieving an average perplexity reduction of 1.82/1.47/0.29 and 80.87/4606.98/678658.74 compared to OPTQ, LUT-GEMM, and AWQ at 3 and 2 bits, respectively. Evaluating *Ours (Lat.)* with both accuracy and latency metrics as shown in Fig. 6 (c & d), *Ours*

Table 3: Perplexity comparisons of the LLaMA models on WikiText-2. The group size is set to 128 following [44, 34].

LLaMA (PPL ↓)	Bits	LLaMA-1	LLaMA-2			LLaMA-3	
		7B	7B	13B	70B	8B	70B
FP16	16	5.68	5.47	4.88	3.32	6.14	2.86
OPTQ [17]	3	8.81	6.43	5.48	3.88	13.69	4.91
LUT-GEMM [44]	3	7.18	7.02	5.89	4.01	11.10	5.92
AWQ [34]	3	6.35	6.24	5.32	3.74	8.15	4.69
Ours (Acc.)	3	**6.04**	**5.89**	**5.16**	**3.64**	**7.20**	**4.35**
OPTQ [17]	2	68.60	19.92	12.75	6.82	398.0	26.65
LUT-GEMM [44]	2	303.00	2242.0	2791.0	136.4	19096	3121
AWQ [34]	2	2.6e5	2.2e5	1.2e5	7.2e4	1.7e6	1.7e6
Ours (Acc.)	2	**7.98**	**8.51**	**6.77**	**4.72**	**12.07**	**7.51**

(Lat.) demonstrates better accuracy-latency tradeoffs. It achieves 1.1~1719987.6 perplexity reduction at comparable latency or 19.9%~65.0% latency reductions and 28.4%~89.9% energy savings at similar or even lower perplexity. Complete quantitative data on accuracy, latency, and energy are provided in Appendix B.

Results on Gemma/Mistral/Bloom Models. We also evaluate ShiftAddLLM on Gemma [38], Mistral [29], and Bloom [45] models, which are among the most popular open-source LLMs and Mixture-of-Expert (MoE) models. As shown in Tab. 4, *Ours (Acc.)* achieves perplexity reductions of 11.12/29.4 for Gemma-2B, 1.67/16.76 for Mistral-

Table 4: Results on Gemma/Mistral/Bloom models.

PPL (↓)	Bits	Gemma-2B	Mistral-7B	Bloom-3B	Bloom-7B
FP16	16	13.88	5.25	13.48	11.37
OPTQ	3	26.08	7.27	17.40	13.47
LUT-GEMM	3	44.36	22.36	21.03	17.29
Ours (Acc.)	3	**14.96**	**5.60**	**14.10**	**11.71**

7B, and 3.30/6.93 and 1.76/5.58 for BLOOM-3B/7B, respectively, compared to OPTQ and LUT-GEMM. As shown in Fig. 6 (e), *Ours (Lat.)* shows better accuracy-latency tradeoffs, e.g., achieving 9.56 perplexity reduction and 11% latency reductions over the OTPQ baseline on Gemma models. Tehse results on five LLM families consistently validate the effectiveness of our ShiftAddLLM.

Table 6: Perplexity and latency results of our mixed bit allocation.

Methods	Bits	PPL (↓)						Latency (ms)					
		125M	350M	1.3B	2.7B	6.7B	13B	125M	350M	1.3B	2.7B	6.7B	13B
Ours (Lat.)	2	712.55	445.78	40.28	50.95	18.56	14.76	6.3	12.4	12.3	16.9	16.9	20.9
Ours (Mixed)	2.2	435.84	279.19	27.37	31.97	17.99	13.79	6.3	12.6	12.5	16.8	16.7	21.0

Figure 8: Energy breakdown for OPT-66B and LLaMA-70B models using an Eyeriss accelerator.

Zero-shot Downstream Tasks. We extend our evaluation to zero-shot downstream datasets for a more comprehensive assessment. As shown in Tab. 5, *Ours (Acc.)* consistently improves performance over previous SOTA baselines, achieving an average accuracy gain of 13.59/13.61 and 2.61/2.39 over OPTQ and LUT-GEMM baselines at 3 bits when evaluated on OPT-66B and LLaMA-2-70B, respectively. These experiments demonstrate that our method not only reduce perplexity but also improves downstream task accuracy.

GPU Memory Savings. Our ShiftAddLLM also reduces GPU memory usage. For OPT-66B, our method saves 81% and 87% memory costs over FP16 at 3 (23GB vs. 122GB) and 2 bits (16GB vs. 122GB), respectively. For LLaMA-2-70B, it saves 80% and 87% memory costs at 3 (25GB vs. 128GB) and 2 bits (17GB vs. 128GB), respectively.

Results of Mixed Bit Allocation. We evaluate our mixed bit allocation strategy (see Sec. 4.3) and compare *Ours (Mixed)* with *Ours (Lat.)*. As shown in Tab. 6, *Ours (Mixed)* further reduces the perplexity by an average of 79.45 for OPT model families under comparable or even less latency.

5.3 Ablation Studies of ShiftAddLLM

Visualization of Mixed Bit Allocation. We visualize the bit allocations after applying our automated bit allocation strategy with an average budget of 2.2 bits (Fig. 7). The allocation pattern correlates with the sensitivity to reparameterization identified in Sec. 4.3 and shown in Fig. 4. For instance, later blocks, which experience more quantization or reparameterization errors, receive more bits. The K linear layers and the first MLP (FC1) in each block are also allocated hi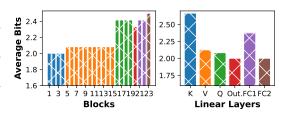gher bits. This visualization confirms that our strategy effectively adjusts bits according to reparameterization errors.

Figure 7: Visualizing the average bit allocation.

Performance and Energy Breakdown. To examine the contribution of each proposed technique, we conducted ablation studies on OPT-6.7B/13B models. As shown in Tab. 7, the vanilla ShiftAddLLM (Sec. 4.1) suffers from a significant perplexity increase with 2-bit reparameterization. Our multi-objective optimization (Sec. 4.2) reduces perplexity by an average of 3.9e4, and the mixed bit allocation strategy (Sec. 4.3) further reduces perplexity by 0.77, maintaining comparable latency. These experiments validate the ef-

Table 7: Performance breakdown analysis.

OPT w/ Sec.	Bits	PPL		Latency (ms)	
		6.7B	13B	6.7B	13B
4.1	2	6.4e4	1.5e4	16.5	20.1
4.1&4.2	2	18.56	14.76	16.9	20.9
4.1&4.2& 4.3	2.2	**17.99**	**13.79**	16.7	21.0

fectiveness of each component in ShiftAddLLM. In addition, profiling the two largest models on an Eyeriss accelerator illustrates the energy breakdown of the original LLMs and ShiftAddLLMs. As shown in Fig. 8, ShiftAddLLM reduces 87.2% for OPT-66B and 86.0% for LLaMa-2-70B, with shift-and-add leading to 89.7% and 89.9% energy reduction compared to original multiplications.

5.4 Limitation and Broader Societal Impact Discussion

We demonstrated the accuracy and efficiency of post-training shift-and-add reparameterization of LLMs using multi-objective optimization and automated bit allocation, addressing the challenge of efficient LLM serving. However, achieving GPU speedup relied on BCQ kernels and the compatible Ours (Lat.) with a block-wise scaling factor design. While Ours (Acc.) with a column-wise design delivers high accuracy, we lack the fast CUDA kernel required for similar speedups.

6 Conclusion

We propose accelerating pretrained LLMs through post-training shift-and-add reparameterization, creating efficient multiplication-free models. Our method reparameterizes weight matrices into binary matrices with group-wise scaling factors, transforming multiplications into shifts and adds. To mitigate accuracy loss, we introduce a multi-objective optimization strategy that minimizes weight and activation reparameterization errors. Additionally, we develop an automated bit allocation strategy based on layer sensitivity to further improve the accuracy-efficiency tradeoff. Extensive results across various LLM families and tasks validate the effectiveness of ShiftAddLLM. This work opens a new perspective on designing efficient LLM serving systems through *post-training* optimization.

Acknowledgments and Disclosure of Funding

References

[1] Ardavan Afshar, Ioakeim Perros, Evangelos E Papalexakis, et al. COPA: Constrained PARAFAC2 for sparse & large datasets. In CIKM, 2018.

[2] Meta AI. LLaMA 3. `https://github.com/meta-llama/llama3`, 2024.

[3] Rohan Anil, Sebastian Borgeaud, et al. Gemini: A Family of Highly Capable Multimodal Models. arXiv preprint arXiv:2312.11805, 2023.

[4] Michael Boratko, Harshit Padigela, Divyendra Mikkilineni, et al. A Systematic Classification of Knowledge, Reasoning, and Context within the ARC Dataset. arXiv preprint arXiv:1806.00358, 2018.

[5] Diogo Brito, Taimur G Rabuske, Jorge R Fernandes, et al. Quaternary logic lookup table in standard CMOS. TVLSI, 2014.

[6] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, et al. QuIP: 2-Bit Quantization of Large Language Models With Guarantees. NeurIPS, 2024.

[7] Hanting Chen, Yunhe Wang, Chunjing Xu, et al. AdderNet: Do We Really Need Multiplications in Deep Learning? In CVPR, 2020.

[8] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, et al. Eyeriss: An Energy-efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. JSSCC, 2016.

[9] Christopher Clark, Kenton Lee, Ming-Wei Chang, et al. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. arXiv preprint arXiv:1905.10044, 2019.

[10] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, et al. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. arXiv preprint arXiv:1602.02830, 2016.

[11] Ido Dagan, Dan Roth, Fabio Zanzotto, et al. Recognizing Textual Entailment: Models and Applications. Springer Nature, 2022.

[12] Tri Dao, Dan Fu, Stefano Ermon, et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. NeurIPS, 2022.

[13] Tim Dettmers and Luke Zettlemoyer. The Case for 4-bit Precision: k-bit Inference Scaling Laws. In ICML, 2023.

[14] Tim Dettmers, Mike Lewis, Younes Belkada, et al. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. NeurIPS, 2022.

[15] Mostafa Elhoushi, Zihao Chen, Farhan Shafiq, et al. DeepShift: Towards Multiplication-Less Neural Networks. In CVPR, 2021.

[16] Elias Frantar and Dan Alistarh. Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning. NeurIPS, 2022.

[17] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, et al. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In ICLR, 2022.

[18] Amir Gholami, Zhewei Yao, Sehoon Kim, et al. AI and Memory Wall. IEEE Micro Journal, 2024.

[19] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, et al. The Unreasonable Ineffectiveness of the Deeper Layers. arXiv preprint arXiv:2403.17887, 2024.

[20] Yiwen Guo, Anbang Yao, Hao Zhao, et al. Network Sketching: Exploiting Binary Structure in Deep CNNs. In CVPR, 2017.

[21] Bah-Hwee Gwee, Joseph S Chang, Yiqiong Shi, et al. A Low-Voltage Micropower Asynchronous Multiplier With Shift–Add Multiplication Approach. IEEE Transactions on Circuits and Systems I: Regular Papers, 2008.

[22] Song Han, Xingyu Liu, Huizi Mao, et al. EIE: Efficient Inference Engine on Compressed Deep Neural Network. ACM SIGARCH Computer Architecture News, 2016.

[23] Simla Burcu Harma, Ayan Chakraborty, Elizaveta Kostenok, Danila Mishin, Dongho Ha, Babak Falsafi, Martin Jaggi, Ming Liu, Yunho Oh, Suvinay Subramanian, and Amir Yazdanbakhsh. Effective Interplay between Sparsity and Quantization: From Theory to Practice. arXiv preprint arXiv:2405.20935, 2024.

[24] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal Brain Surgeon and General Network Pruning. In IEEE international conference on neural networks, 1993.

[25] Mark Horowitz. 1.1 Computing's Energy Problem (and what we can do about it). In ISSCC, 2014.

[26] Wei Huang, Yangdong Liu, Haotong Qin, et al. BiLLM: Pushing the Limit of Post-Training Quantization for LLMs. arXiv preprint arXiv:2402.04291, 2024.

[27] Yongkweon Jeon, Baeseong Park, Se Jung Kwon, et al. BiQGEMM: Matrix Multiplication with Lookup Table For Binary-Coding-based Quantized DNNs. In SC, 2020.

[28] Yongkweon Jeon, Chungman Lee, Eulrang Cho, et al. Mr.BiQ: Post-Training Non-Uniform Quantization based on Minimizing the Reconstruction Error. In CVPR, 2022.

[29] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, et al. Mistral 7B. arXiv preprint arXiv:2310.06825, 2023.

[30] Felix Juefei-Xu, Vishnu Naresh Boddeti, and Marios Savvides. Local Binary Convolutional Neural Networks. In CVPR, 2017.

[31] Se Jung Kwon, Dongsoo Lee, Yongkweon Jeon, et al. Post-Training Weighted Quantization of Neural Networks for Language Models. https://openreview.net/forum?id=2Id6XxTjz7c, 2021.

[32] Xinlin Li, Bang Liu, Rui Heng Yang, et al. DenseShift: Towards Accurate and Efficient Low-Bit Power-of-Two Quantization. In ICCV, 2023.

[33] Yuhang Li, Xin Dong, and Wei Wang. Additive Powers-of-Two Quantization: An Efficient Non-uniform Discretization for Neural Networks. arXiv preprint arXiv:1909.13144, 2019.

[34] Ji Lin, Jiaming Tang, Haotian Tang, et al. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. arXiv preprint arXiv:2306.00978, 2023.

[35] Zechun Liu, Barlas Oguz, Changsheng Zhao, et al. LLM-QAT: Data-free Quantization Aware Training for Large Language Models. arXiv preprint arXiv:2305.17888, 2023.

[36] Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the Structural Pruning of Large Language Models. NeurIPS, 2023.

[37] Stephen Merity, Caiming Xiong, James Bradbury, et al. Pointer Sentinel Mixture Models. In ICLR, 2017.

[38] Thomas Mesnard, Cassidy Hardin, et al. Gemma: Open Models Based on Gemini Research and Technology. arXiv preprint arXiv:2403.08295, 2024.

[39] Nasrin Mostafazadeh, Michael Roth, Annie Louis, et al. LSDSem 2017 Shared Task: The Story Cloze Test. In Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics, 2017.

[40] Mohammad Mozaffari, Amir Yazdanbakhsh, Zhao Zhang, and Maryam Mehri Dahnavi. SLoPe: Double-Pruned Sparse Plus Lazy Low-rank Adapter Pretraining of LLMs. arXiv preprint arXiv:2405.16325, 2024.

[41] NVIDIA Corporation. NVIDIA A100 Tensor Core GPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf, 2020. Datasheet.

[42] OpenAI. ChatGPT: Language Model for Dialogue Generation. https://www.openai.com/chatgpt/, 2023. Website.

[43] OpenAI. GPT-4 Technical Report. arXiv preprint arXiv:2303.08774, 2023.

[44] Gunho Park, Baeseong Park, Minsub Kim, et al. LUT-GEMM: Quantized Matrix Multiplication based on LUTs for Efficient Inference in Large-Scale Generative Language Models. arXiv preprint arXiv:2206.09557, 2022.

[45] Teven Le Scao, Angela Fan, Christopher Akiki, et al. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. arXiv preprint arXiv:2211.05100, 2022.

[46] Olivier Sentieys. Approximate Computing for DNN. In CSW 2021-HiPEAC Computing Systems Week, 2021.

[47] Xuan Shen, Zhenglun Kong, Changdi Yang, et al. EdgeQAT: Entropy and Distribution Guided Quantization-Aware Training for the Acceleration of Lightweight LLMs on the Edge. arXiv preprint arXiv:2402.10787, 2024.

[48] Han Shu, Jiahao Wang, Hanting Chen, et al. Adder Attention for Vision Transformer. NeurIPS, 2021.

[49] Mingjie Sun, Zhuang Liu, Anna Bair, et al. A Simple and Effective Pruning Approach for Large Language Models. arXiv preprint arXiv:2306.11695, 2023.

[50] Sandeep Tata and Jignesh M Patel. PiQA: An Algebra for Querying Protein Data Sets. In SSDBM, 2003.

[51] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971, 2023.

[52] Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv preprint arXiv:2307.09288, 2023.

[53] Ethan Waisberg, Joshua Ong, Mouayad Masalkhi, et al. Google's AI chatbot "Bard": A Side-by-Side Comparison with ChatGPT and its Utilization in Ophthalmology. Eye, 2023.

[54] Guangting Wang, Yucheng Zhao, Chuanxin Tang, et al. When Shift Operation Meets Vision Transformer: An Extremely Simple Alternative to Attention Mechanism. In AAAI, 2022.

[55] Yunhe Wang, Mingqiang Huang, Kai Han, et al. AdderNet and its Minimalist Hardware Design for Energy-Efficient Artificial Intelligence. arXiv preprint arXiv:2101.10015, 2021.

[56] Bichen Wu, Alvin Wan, Xiangyu Yue, et al. Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions. In CVPR, 2018.

[57] Guangxuan Xiao, Ji Lin, Mickael Seznec, et al. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In ICML, 2023.

[58] Chen Xu, Jianqiang Yao, Zhouchen Lin, et al. Alternating Multi-bit Quantization for Recurrent Neural Networks. arXiv preprint arXiv:1802.00150, 2018.

[59] Yixing Xu, Chang Xu, Xinghao Chen, et al. Kernel Based Progressive Distillation for Adder Neural Networks. In NeurIPS, 2020.

[60] Ping Xue and Bede Liu. Adaptive Equalizer Based on a Power-Of-Two-Quantized-LMF Algorithm. IEEE transactions on acoustics, speech, and signal processing, 1986.

[61] Songlin Yang, Bailin Wang, Yikang Shen, et al. Gated Linear Attention Transformers with Hardware-Efficient Training. arXiv preprint arXiv:2312.06635, 2023.

[62] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, et al. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. NeurIPS, 2022.

[63] Haoran You, Xiaohan Chen, Yongan Zhang, et al. ShiftAddNet: A Hardware-Inspired Deep Network. NeurIPS, 2020.

[64] Haoran You, Baopu Li, Shi Huihong, et al. ShiftAddNAS: Hardware-Inspired Search for More Accurate and Efficient Neural Networks. In ICLR, 2022.

[65] Haoran You, Yichao Fu, Zheng Wang, et al. When Linear Attention Meets Autoregressive Decoding: Towards More Effective and Efficient Linearized Large Language Models. In ICML, 2024.

[66] Haoran You, Huihong Shi, Yipin Guo, et al. ShiftAddViT: Mixture of multiplication primitives towards efficient vision transformer. NeurIPS, 2024.

[67] Susan Zhang, Stephen Roller, Naman Goyal, et al. OPT: Open Pre-trained Transformer Language Models. arXiv preprint arXiv:2205.01068, 2022.

[68] Yang Zhao, Chaojian Li, Yue Wang, et al. DNN-Chip Predictor: An Analytical Performance Predictor for DNN Accelerators with Various Dataflows and Hardware Architectures. In ICASSP, 2020.

[69] Rui-Jie Zhu, Yu Zhang, Ethan Sifferman, Tyler Sheaves, Yiqiao Wang, Dustin Richmond, Peng Zhou, and Jason K Eshraghian. Scalable matmul-free language modeling. arXiv preprint arXiv:2406.02528, 2024.

A Complete Accuracy & Latency & Energy Data for OPT Models

We supply the complete quantitative accuracy, latency, and energy data measured on the OPT model family in Tab. 8, 9, and 10, respectively.

Table 8: Perplexity comparisons of the OPT models on WikiText-2. Note that we set the group size of all methods as the number of columns following the setting of OPTQ [17] for a fair comparison.

OPT (PPL ↓)	Bits	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B
FP16	16	27.65	22.00	14.62	12.47	10.86	10.13	9.56	9.34
OPTQ [17]	3	53.85	33.79	20.97	16.88	14.86	11.61	10.27	14.16
LUT-GEMM [44]	3	60.00	42.32	49.10	17.55	17.44	12.50	139.90	100.33
AWQ [34]	3	54.75	35416.00	24.60	39.01	16.47	16.53	31.01	5622.00
Ours (Lat.)	3	56.96	28.72	19.69	15.28	11.80	10.70	9.89	9.62
OPTQ [17]	2	2467.50	10433.30	4737.05	6294.68	442.63	126.09	71.70	20.91
LUT-GEMM [44]	2	4844.32	2042.90	3851.50	616.30	17455.52	4963.27	7727.27	6246.00
AWQ [34]	2	3514.61	18313.24	9472.81	22857.70	8168.30	5014.92	7780.96	103843.84
Ours (Lat.)	2	712.55	445.78	40.28	50.95	18.56	14.76	12.55	12.20
Ours (Mixed)	2.2	435.84	279.19	27.37	31.97	17.99	13.79	11.62	11.17

Table 9: A100 GPU latency comparisons on the OPT model family.

OPT Latency (ms)	Bits	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B
FP16	16	7.8	15.1	16.7	20.9	22.2	29.5	51.7	OOM
OPTQ [17]	3	8.3	15.9	15.0	21.5	21.1	26.4	30.1	51.5
LUT-GEMM [44]	3	6.3	11.7	12.6	15.5	17.0	19.5	23.7	39.5
AWQ [34]	3	6.2	12.1	12.3	16.3	16.3	20.0	24.5	40.9
Ours (Lat.)	3	6.4	13.3	12.6	16.6	16.9	20.8	30.7	54.1
OPTQ [17]	2	8.2	16.1	15.9	19.7	19.9	24.7	31.5	50.4
LUT-GEMM [44]	2	6.2	11.8	11.8	15.7	15.7	19.8	23.6	33.2
AWQ [34]	2	6.2	12.1	12.3	16.3	16.3	20.0	24.5	40.9
Ours (Lat.)	2	6.3	12.4	12.3	16.9	16.9	20.9	25.4	42.9
Ours (Mixed)	2.2	6.3	12.6	12.5	16.8	16.7	21.0	27.1	45.7

* Note that we use AWQ's open-sourced INT4 kernel for measuring its latency.

Table 10: Energy comparisons on the OPT model family.

OPT Energy (J)	Bits	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B
FP16	16	29.26	83.72	310.33	625.80	1573.41	3036.99	7088.39	15539.87
OPTQ [17]	3	13.77	28.63	90.12	167.37	399.28	745.37	1695.45	3658.17
LUT-GEMM [44]	3	12.83	25.19	75.73	137.08	321.54	592.31	1332.95	2858.87
Ours (Lat.)	3	11.68	21.13	59.59	103.53	235.45	424.58	938.98	1990.17
OPTQ [17]	2	12.58	24.42	73.27	132.30	309.50	570.06	1283.04	2749.32
LUT-GEMM [44]	2	12.48	23.96	70.90	127.07	295.77	542.28	1215.36	2599.77
Ours (Lat.)	2	11.41	20.17	55.80	95.67	215.27	385.31	846.54	1786.62
Ours (Mixed)	2.2	11.45	20.33	56.43	96.98	218.64	391.86	861.95	1820.55

B   Complete Accuracy & Latency & Energy Data for LLaMA Models

We supply the complete quantitative accuracy, latency, and energy data measured on the LLaMA model family in Tab. 11, 12, and 13, respectively.

Table 11: Perplexity comparisons of the LLaMA models on WikiText-2.

LLaMA (PPL ↓)	Bits	LLaMA-2			LLaMA-3	
		7B	13B	70B	8B	70B
FP16	16	5.47	4.88	3.32	6.14	2.86
OPTQ [17]	3	6.43	5.48	3.88	13.69	4.91
LUT-GEMM [44]	3	7.02	5.89	4.01	11.10	5.92
AWQ [34]	3	6.24	5.32	3.74	8.15	4.69
Ours (Lat.)	3	6.04	5.33	3.72	7.71	4.66
OPTQ [17]	2	19.92	12.75	6.82	398.0	26.65
LUT-GEMM [44]	2	2242.0	2791.0	136.4	19096	3121
AWQ [34]	2	2.22e5	1.22e5	7.24e4	1.71e6	1.72e6
Ours (Lat.)	2	9.58	12.57	5.71	34.4	12.4

* Note that the group size is set to 128 following [44, 34].

Table 12: A100 GPU latency comparisons of the LLaMA models.

LLaMA Latency (ms)	Bits	LLaMA-2			LLaMA-3	
		7B	13B	70B	8B	70B
FP16	16	32.6	43.1	OOM	38.8	OOM
OPTQ [17]	3	31.1	42.2	81.9	36.2	90.7
LUT-GEMM [44]	3	27.4	34.7	72.6	31.7	77.5
AWQ [34]	3	25.4	31.8	68.0	28.5	67.7
Ours (Lat.)	3	26.7	33.8	70.9	31.4	72.9
OPTQ [17]	2	34.2	38.8	82.5	36.8	91.2
LUT-GEMM [44]	2	27.5	33.3	71.0	31.7	77.2
AWQ [34]	2	25.4	31.8	68.0	28.5	67.7
Ours (Lat.)	2	27.7	33.9	72.1	31.9	78.3
Ours (Mixed)	2.2	27.2	34.3	75.1	30.1	76.4

Table 13: Energy comparisons of the LLaMA models.

LLaMA Energy (J)	Bits	LLaMA-2			LLaMA-3	
		7B	13B	70B	8B	70B
FP16	16	1563.44	3040.26	18482.5	1776.05	16445.98
OPTQ [17]	3	383.40	728.98	4297.33	504.07	3972.72
LUT-GEMM [44]	3	305.06	574.71	3349.01	419.64	3139.34
Ours (Lat.)	3	218.59	405.53	2309.87	326.47	2225.71
OPTQ [17]	2	293.15	552.20	3212.56	406.81	3018.87
LUT-GEMM [44]	2	279.20	524.16	3037.94	391.74	2865.81
Ours (Lat.)	2	198.33	365.85	2065.90	304.59	2011.15
Ours (Mixed)	2.2	201.69	372.40	2099.53	306.69	2066.64

C   Ablation Studies on Multi-Objective Optimization

We conduct ablation studies on different optimization objectives. As shown in Tab. 14, our multi-objective optimization demonstrates superior performance in both column-wise and block-wise scaling factor formats. It achieves average perplexity reductions of 123.25, 2.22, and 403.18 compared to the weight-only objective, activation-only objective, and the vanilla combination of both weight and activation objectives, respectively. These experiments validate the effectiveness of our multi-objective optimization approach.

Table 14: Ablation studies on various optimization objectives.

OPT PPL	13B	30B	66B
Wei. Obj.	13.8	222.6	163.2
Act. Obj.	11.7	10.5	14.3
Wei. + Act.	45.0	16.3	1178.1
Ours (Col.-wise)	**10.4**	**9.6**	**9.4**
Ours (Blk.-wise)	**10.8**	**9.9**	**9.6**