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 reductions of 5.6 and 22.7 points at comparable or lower latency compared to the most competitive quantized LLMs at 3- and 2-bit precision, 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 [46, 47, 59, 3, 74, 57, 58, 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 [38] and performs 10^{15} floating-point operations (FLOPs) for a single forward pass [19]. Previous efforts to improve LLM efficiency have focused on pruning [40, 55, 20, 24, 44], quantization [63, 38, 18, 48], and attention optimization [12, 71, 67]. However, these methods still rely on costly multiplication operations in both the attention and MLP layers.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Table 1: Hardware cost under 45nm CMOS [27, 69, 23, 50, 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 [66, 22] can offer up to $^{3.1}/_{0.1} = 31\times$ energy and $^{3495}/_{137} \approx 26\times$ area reductions (see Tab. 1). This hardware-inspired approach can lead to efficient and fast implementations, as shown by previous research on ShiftAddNet [69, 70, 72]. Unlike previous 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 [69, 72] can result in nontrivial quantization errors, requiring 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 model 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 reductions 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) [39, 52] and post-training quantization (PTQ) [18, 38, 63, 15]. 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 [63, 15, 68], often limited to 8 bits (W8A8) as lower bit representations can significantly reduce accuracy; and ***(2)*** lower bit weight-only quantization [18, 48, 14, 28, 6], which quantizes LLM weights to lower bits while keeping activations in a FP16 format.

This approach alleviates memory bottlenecks associated with the vast parameters of LLMs. For instance, GPTQ [18] uses gradient-based weight quantization and develops `INT3/4` kernels to reduce data movements, and LUT-GEMM [48] 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, 32] binarize weights and activations, while shift-based networks use spatial shifts [62] or bitwise shifts [16] to substitute for multiplications. AdderNet [7, 65, 61] replaces multiplications with additions, albeit with a small accuracy drop. ShiftAddNet [69] reparameterizes CNNs with cascaded shift and add layers. These techniques have been adapted to Transformers. BiLLM [28] introduces binary LLMs, while [54] and [60] extend the addition or shift concepts to the attention mechanisms, respectively. ShiftAddViT [72] reparameterizes pretrained Vision Transformers (ViTs) with shifts and adds. Contemporary work MatMul-free LM [76] leverages additive operators and Hadamard products for multiplication-free language model training, relying on FPGAs for speedups. Compared to closely related works like ShiftAddNet [69] and MatMul-free LM [76], which requires training from scratch, and ShiftAddViT [72], 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 [64] 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 [64, 30, 33], as shown in Alg. 1. Initially, we use the greedy method [21] 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 [64]

Input: Full-precision weight $\mathbf{w} \in \mathbb{R}^n$, bit-width q, alternating cycles T
Output: $\alpha_i^*, \mathbf{b}_i^* \in \{-1, 1\}^{m\times n}$
Function MULTI-BIT BCQ($\mathbf{w}, q, T$)
  $\{\alpha_i, \mathbf{b}_i\}_{i=1}^{q} \leftarrow$ GREEDY($\mathbf{w}$)
  for $t \leftarrow 1$ to T **do**
    $\{\alpha_i\}_{i=1}^{q} \leftarrow \text{LS}(\mathbf{B}, \mathbf{w})$
    $\{\mathbf{b}_i\}_{i=1}^{q} \leftarrow \text{BS}(\alpha_1, \ldots, \alpha_q, \mathbf{w})$
  end for
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) [21] 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)*** [64].

Such BCQ can support both uniform and non-uniform quantization formats by adjusting the scaling factors and biases accordingly [48]. 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

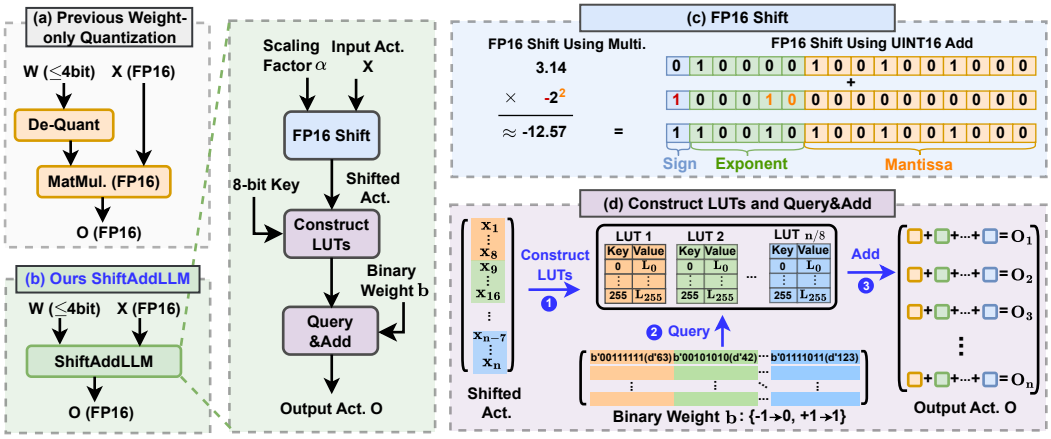

Figure 1: Illustration of our proposed post-training reparameterization for ShiftAddLLM.

equivalent to multiplying by powers of two, offer a non-uniform quantization solution and can result in significant savings. For example, we tested matrix multiplication from one MLP layer of OPT-66B between weight $W \in \mathbb{R}^{9216\times 36884}$ and activation $A \in \mathbb{R}^{1\times 9216}$ using FP16 MACs and our 3-bit ShiftAddLLM. Energy consumption was 80.36J vs. 9.77J, achieving 87.8% savings with our method. Both primitives have inspired many innovations in efficient model innovations [7, 16, 69, 72].

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 [18], 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 [48, 29], 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) [37], 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 [36], as also illustrated in Fig. 1 (c); (2) Queries and

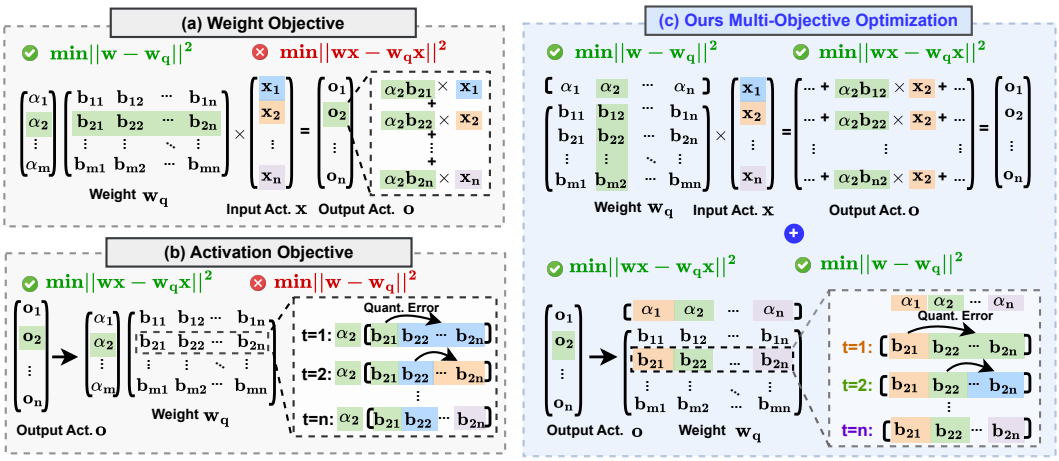

Figure 2: Illustration of our proposed multi-objective optimization framework.

adds 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. Here every eight grouped binary weights form an 8-bit key. Suppose the shifted activation is an n-dimensional vector. In that case, we will get $n/8$ LUTs, where the grouped binary weights are used as keys, and the precomputed partial sums are stored as values. 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 in FP16 format. Such LUTs are well supported by existing GPU kernels [48, 29]. The reparameterization can be applied to all weights in pretrained LLMs in a *post-training* manner, replacing costly multiplications with efficient hardware operations.

Takeaway. 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 [63, 38], rescale the weight quantization error differently, causing significant divergence in the output activation. For example, LUT-GEMM [48] adopts this weight objective. On the other hand, the "*activation objective*" (see Fig. 2 (b)) minimizes the output activation error, i.e., $\|\mathbf{WX} - \mathbf{W}_q\mathbf{X}\|$, 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 [18] 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 [48] 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 [18].

Accuracy vs. Latency Trade-offs. The column-wise scaling factor design significantly boosts accuracy after reparameterization. However, it does not fully leverage BCQ [48, 29], which process eight elements per row of weights in parallel as LUT keys, resulting in latency overhead for models with ≥30B parameters. For example, testing on the OPT-30B [74] model and WikiText-2 dataset [41] showed $(16.3 - 9.6) = 6.7$ perplexity reduction but with a $^{(44.1-33.2)}/_{44.1} \approx 24.7\%$ latency overhead, as shown in Fig. 3 (b).

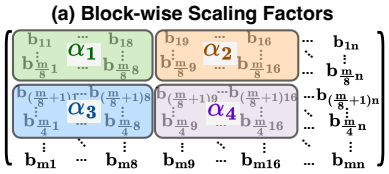

Figure 3: (a) the block-wise scaling factors and (b) the comparison among different designs on OPT-30B [74].

To address this, we propose a block-wise scaling factor design that groups 8 columns and 1/8 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.

Takeaway. 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 [38].

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, Query/Key (Q/K) 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.

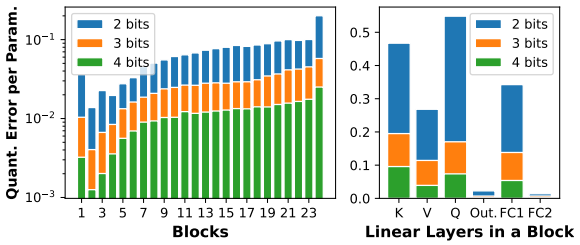

Figure 4: Sensitivity analysis on OPT-1.3B [74].

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:

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

where the importance score (IS) is inspired by Optimal Brain Compression [25, 17, 18] and is correlated to the increase in the quadratic reconstruction error $\|\mathbf{W}\mathbf{X} - \mathbf{W}_q\mathbf{X}\|^2$ after reparameterizing the weights, i.e., IS ↑, error increases ↓. 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

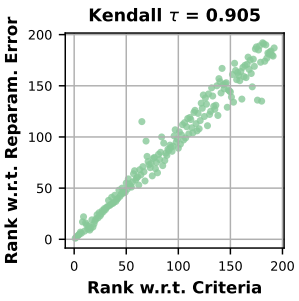

Figure 5: Rank comparisons.

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 [18] 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 [18]	3	53.85	33.79	20.97	16.88	14.86	11.61	10.27	14.16
LUT-GEMM [48]	3	60.00	42.32	49.10	17.55	17.44	12.50	139.90	100.33
AWQ [38]	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 [18]	2	2467.50	10433.30	4737.05	6294.68	442.63	126.09	71.70	20.91
LUT-GEMM [48]	2	4844.32	2042.90	3851.50	616.30	17455.52	4963.27	7727.27	6246.00
AWQ [38]	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**

to the actual reparameterization 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 $C_{i,b}$.

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

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

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.

Takeaway. 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 [74], LLaMA-1/2/3 [58, 2], Gemma [42], Mistral [31], and Bloom [49]. ***Tasks and Datasets.*** We evaluate all five LLMs on the commonly adopted language modeling task using the WikiText-2 [41] dataset for perplexity measurement. Additionally, we extend the evaluation of the two largest models, OPT-66B and LLaMA-2-70B, to eight downstream tasks for zero-shot accuracy evaluation. These tasks include ARC (Challenge/Easy) [4], BoolQ [9], Copa [1], PIQA [56], RTE [11], StoryCloze [43], and MMLU [26]. ***Baselines.*** We consider four SOTA LLM quantization methods: OPTQ [18], LUT-GEMM [48], QuIP [6], and AWQ [38]. ***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 eight downstream tasks. For efficiency, we measure the latency on a single A100-80GB GPU (PCIe) [45] and estimate the energy costs using an Eyeriss-like hardware accelerator [8, 75], which calculates not only computational but also data movement energy (within 18% of the differences with Eyeriss's chip measurement results as claimed).

5.2 ShiftAddLLM over SOTA LLM Quantization Baselines

Results on OPT Models. To evaluate the effectiveness of our ShiftAddLLM, we compare against four SOTA LLM quantization baselines: OPTQ [18], LUT-GEMM [48], QuIP [6], and AWQ [38]. Using the OPT model family [74] and the WikiText-2 dataset [41], 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,

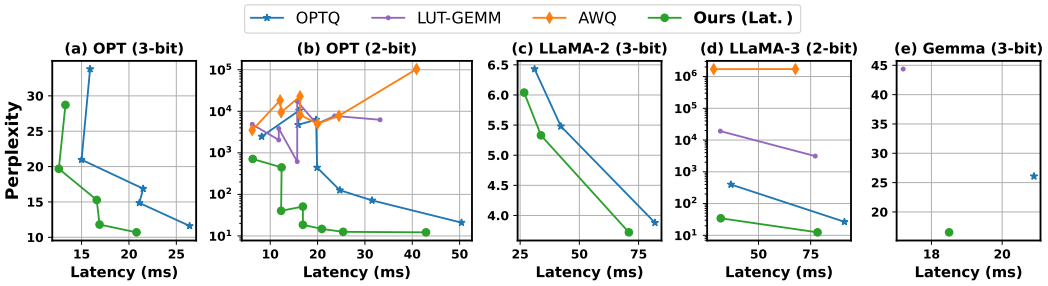

Figure 6: Accuracy-latency tradeoff comparisons on the OPT, LLaMA-2/3, and Gemma models.

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, with a perplexity reduction of 0.91∼103830.45 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 is provided in Appendix A.

Results on LLaMA Models. We further evaluate ShiftAddLLM on LLaMA models [57, 58, 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 (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.

Table 3: Perplexity comparisons of the LLaMA models on WikiText-2. The group size is set to 128 following [48, 38].

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 [18]	3	8.81	6.43	5.48	3.88	13.69	4.91
LUT-GEMM [48]	3	7.18	7.02	5.89	4.01	11.10	5.92
AWQ [38]	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 [18]	2	68.60	19.92	12.75	6.82	398.0	26.65
LUT-GEMM [48]	2	303.00	2242.0	2791.0	136.4	19096	3121
AWQ [38]	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**

Results on Gemma/Mistral/Bloom Models. We also evaluate ShiftAddLLM on Gemma [42], Mistral [31], and Bloom [49] 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-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. These results on five LLM families consistently validate the effectiveness of our ShiftAddLLM.

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**

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 13.37/13.19 and 2.55/2.39 over OPTQ and LUT-GEMM baselines at 3 bits when evaluated on OPT-66B and LLaMA-2-70B,

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	MMLU	Mean
OPT-66B	Floating Point	16	37.20	71.25	86	69.82	78.67	77.47	60.65	25.89±0.37	63.37
	OPTQ [18]	3	24.66	48.86	70	52.05	64.47	67.09	53.07	23.98±0.36	50.52
	LUT-GEMM [48]	3	24.15	51.85	81	53.52	61.97	60.60	48.74	23.73±0.36	50.70
	Ours (Acc.)	3	**35.24**	**70.88**	**87**	**72.45**	**77.64**	**77.15**	**63.18**	**27.56±0.38**	**63.89**
LLaMA-2-70B	Floating Point	16	49.57	76.14	90	82.57	80.79	78.61	68.23	65.24±0.37	72.89
	OPTQ [18]	3	45.82	76.34	90	81.74	79.71	77.34	67.51	60.14±0.36	72.33
	LUT-GEMM [48]	3	47.70	76.42	89	80.31	80.20	77.78	68.59	-	-
	Ours (Acc.)	3	**48.38**	**77.06**	**93**	**84.25**	**80.47**	**78.49**	**75.09**	**62.33±0.38**	**74.88**

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

respectively. These experiments demonstrate that our method not only reduces 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. We provide more results in Appendix F to validate the effectiveness of our mixed bit allocation strategy.

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 bit budget of 2.2 (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 higher bits. This visualization confirms that our strategy effectively adjusts bits according to reparameterization errors.

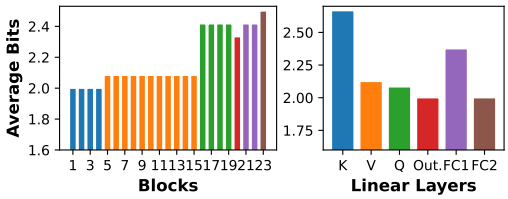

Figure 7: Visualizing the average bit allocation.

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 effectiveness 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.

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

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

As shown in Fig. 8, ShiftAddLLM reduces energy consumption by 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 Discussion on Limitation

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.

5.5 Discussion on Technique Applicability Beyond LLMs

We acknowledge that the idea of shift-and-add reparameterization is general and can be extended to other smaller models like CNNs [69] or ViTs [72]. Meanwhile, this work's implementation is specifically dedicated to large-scale LLMs: It is the first instance of applying the shift-and-add technique at the scale of LLMs with billions of parameters. While many ideas perform well with models having millions of parameters, they often fail to scale effectively. Unlike previous methods that require additional training and do not yield good results for large-scale LLMs, our approach is uniquely tailored for LLMs. We incorporate "post-training" reparameterization and carefully designed scaling factor patterns, enabling multi-objective optimization for LLMs and ensuring superior performance compared to prior quantization methods.

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

This work is supported by the National Science Foundation (NSF) RTML program (Award number: 1937592) and the CoCoSys, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We extend our gratitude towards Mitchelle Rasquinha, and Robert Hundt for reviewing the paper and providing insightful feedback. We also thank the extended team at Google DeepMind who enabled and supported this research direction.

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] Bita Darvish Rouhani, Daniel Lo, Ritchie Zhao, Ming Liu, Jeremy Fowers, Kalin Ovtcharov, Anna Vinogradsky, Sarah Massengill, Lita Yang, Ray Bittner, et al. Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. Advances in neural information processing systems, 33:10271–10281, 2020.

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

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

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

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

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

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

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

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

[22] 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.

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

[24] 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.

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

[26] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300, 2020.

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

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

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

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

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

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

[33] 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.

[34] Jung Hyun Lee, Jeonghoon Kim, Se Jung Kwon, and Dongsoo Lee. Flexround: Learnable rounding based on element-wise division for post-training quantization. In International Conference on Machine Learning, pages 18913–18939. PMLR, 2023.

[35] Jung Hyun Lee, Jeonghoon Kim, June Yong Yang, Se Jung Kwon, Eunho Yang, Kang Min Yoo, and Dongsoo Lee. Lrq: Optimizing post-training quantization for large language models by learning low-rank weight-scaling matrices. arXiv preprint arXiv:2407.11534, 2024.

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

[37] 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.

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

[39] 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.

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

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

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

[43] 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.

[44] 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.

[45] 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.

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

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

[48] 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.

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

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

[51] 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. arXiv preprint arXiv:2308.13137, 2023.

[52] 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.

[53] Huihong Shi, Haoran You, Yang Zhao, Zhongfeng Wang, and Yingyan Lin. Nasa: Neural architecture search and acceleration for hardware inspired hybrid networks. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, pages 1–9, 2022.

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

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

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

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

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

[59] 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.

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

[61] 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.

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

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

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

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

[66] 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.

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

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

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

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

[71] 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.

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

[73] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, et al. Llm inference unveiled: Survey and roofline model insights. arXiv preprint arXiv:2402.16363, 2024.

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

[75] 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.

[76] 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 [18] 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 [18]	3	53.85	33.79	20.97	16.88	14.86	11.61	10.27	14.16
LUT-GEMM [48]	3	60.00	42.32	49.10	17.55	17.44	12.50	139.90	100.33
AWQ [38]	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 [18]	2	2467.50	10433.30	4737.05	6294.68	442.63	126.09	71.70	20.91
LUT-GEMM [48]	2	4844.32	2042.90	3851.50	616.30	17455.52	4963.27	7727.27	6246.00
AWQ [38]	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 [18]	3	8.3	15.9	15.0	21.5	21.1	26.4	30.1	51.5
LUT-GEMM [48]	3	6.3	11.7	12.6	15.5	17.0	19.5	23.7	39.5
AWQ [38]	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 [18]	2	8.2	16.1	15.9	19.7	19.9	24.7	31.5	50.4
LUT-GEMM [48]	2	6.2	11.8	11.8	15.7	15.7	19.8	23.6	33.2
AWQ [38]	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 [18]	3	13.77	28.63	90.12	167.37	399.28	745.37	1695.45	3658.17
LUT-GEMM [48]	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 [18]	2	12.58	24.42	73.27	132.30	309.50	570.06	1283.04	2749.32
LUT-GEMM [48]	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 [18]	3	6.43	5.48	3.88	13.69	4.91
LUT-GEMM [48]	3	7.02	5.89	4.01	11.10	5.92
AWQ [38]	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 [18]	2	19.92	12.75	6.82	398.0	26.65
LUT-GEMM [48]	2	2242.0	2791.0	136.4	19096	3121
AWQ [38]	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 [48, 38].

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 [18]	3	31.1	42.2	81.9	36.2	90.7
LUT-GEMM [48]	3	27.4	34.7	72.6	31.7	77.5
AWQ [38]	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 [18]	2	34.2	38.8	82.5	36.8	91.2
LUT-GEMM [48]	2	27.5	33.3	71.0	31.7	77.2
AWQ [38]	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 [18]	3	383.40	728.98	4297.33	504.07	3972.72
LUT-GEMM [48]	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 [18]	2	293.15	552.20	3212.56	406.81	3018.87
LUT-GEMM [48]	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**

D Impact of Batch Sizes on Throughput

To investigate the impact of batch sizes on the achievable throughput, we have further tested the throughput of our CUDA kernels and end-to-end models with increased batch sizes, as demonstrated in Fig. 9. Our ShiftAddLLM still outperforms all three baselines at a batch size of 8 in terms of accuracy-efficiency trade-offs, achieving on average 3.37×/2.55×/1.39× throughput improvements compared to OPTQ, AWQ, and LUT-GEMM at similar or much better accuracy.

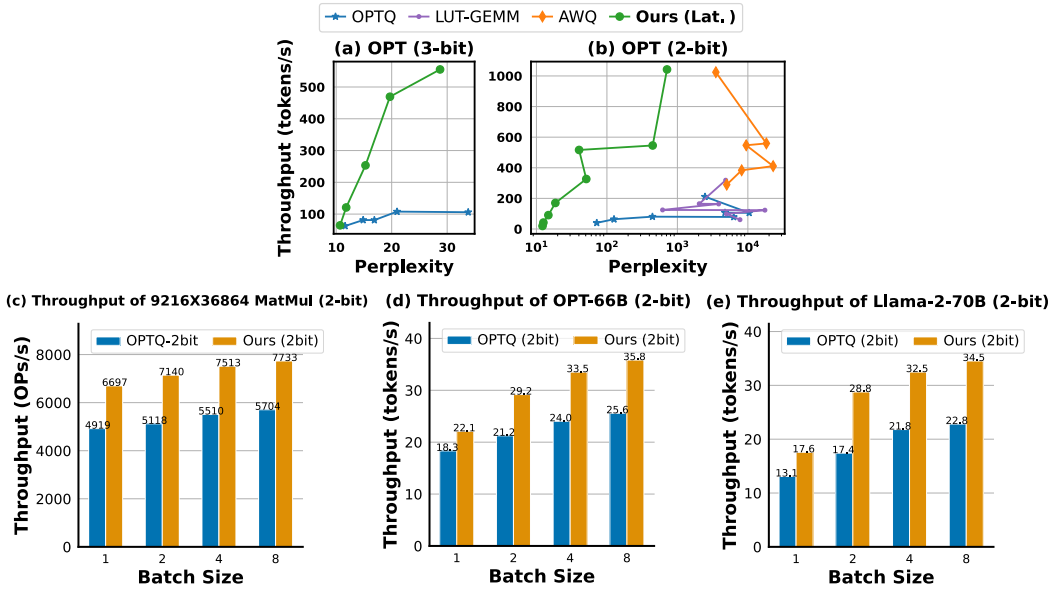

Figure 9: (a-b): Accuracy-throughput tradeoff comparisons among ShiftAddLLM, OPTQ, LUT-GEMM, and AWQ at a batch size of 8. (c) Kernel throughput evaluation under batch sizes of 1, 2, 4, and 8. (d) LLaMA-2-70B end-to-end model throughput evaluation under batch sizes of 1, 2, 4, and 8. (e) OPT-66B end-to-end model throughput evaluation under batch sizes of 1, 2, 4, and 8.

Previously, we assumed a batch size of one for mobile applications where only one user is using the LLM. This assumption also stems from the sequential nature of LLMs during generation, i.e., generating one token at a time based on all previously generated contexts. The assumption of a batch size of 1 is also used in previous literature, such as AWQ, OPTQ, and LUT-GEMM, to measure the latency or throughput for LLM serving.

E Benchmark with More Recent Baselines

We further compare our ShiftAddLLM with recent LLM quantization baselines FlexRound [34] and OmniQuant [51] on OPT and LLaMA models. As shown in Tabs. 15 & 16, our ShiftAddLLM consistently shows better accuracy-efficiency trade-offs, achieving an average of 0.15 (4-bit) / 0.39 (3-bit) and 0.30 (4-bit) / 0.52 (3-bit) perplexity reduction, as compared to FlexRound and OmniQuant, respectively. Note that the baseline results are directly obtained from the original paper and follow-up work LRQ [35]. In addition, we tested OmniQuant at 2 bits ourselves and found it fails for OPT models, whereas ours performs well for OPT models and also achieves an average 1.96 perplexity reduction than OmniQuant on LLaMA at 2 bits.

Table 15: Perplexity comparisons between ShiftAddLLM and OmniQuant using OPT models and LLaMA models on WikiText-2. The group size is set as the length of rows for OPT models and 128 for LLaMA models following baselines.

Method	Bits	OPT								LLaMA-2		
		125M	350M	1.3B	2.7B	6.7B	13B	30B	66B	7B	13B	70B
OmniQuant [51]	4	29.45	23.19	15.04	12.76	11.03	10.30	9.65	-	5.58	4.95	-
Ours (Acc.)	4	**28.72**	**21.59**	**14.98**	**12.65**	**10.95**	**10.20**	**9.63**	-	5.58	4.96	-
OmniQuant [51]	3	35.66	28.2	16.68	13.8	11.65	10.87	10.00	9.83	6.03	5.28	3.78
Ours (Acc.)	3	**31.29**	**24.24**	**21.53**	**13.68**	**11.18**	**10.39**	**9.63**	**9.43**	**5.89**	**5.16**	**3.64**
OmniQuant [51]	2	311.39	186.9	484.51	1.1e6	9.6e5	3.6e4	9.3e3	5.2e3	11.06	8.26	6.55
Ours (Acc.)	2	**51.15**	**40.24**	**29.03**	**20.78**	**13.78**	**12.17**	**10.67**	**10.33**	**8.51**	**6.77**	**4.72**

Table 16: Perplexity comparisons between ShiftAddLLM and FlexRound. The group size of FlexRound is set as the length of rows following the paper.

Method	Bits	LLaMA-2		
		7B	13B	70B
FlexRound [34]	4	5.83	5.01	-
Ours (Acc.)	4	5.58	4.96	-
FlexRound [34]	3	6.34	5.59	3.92
Ours (Acc.)	3	**5.89**	**5.16**	**3.64**

F More Results for Mixed Bit Allocation

To validate the effectiveness and applicability of our automated bit allocation across different LLM models, we evaluated and compared *Ours (Mixed)* with *Ours (Lat.)*. The results are shown in Tab. 17. *Ours (Mixed)* further reduces perplexity by an average of 96.86, 3.23, and 2.63 for OPT, LLaMA, and Gemma models, respectively, under comparable or even less latency. This set of experiments further validates the applicability of our automated bit allocation strategy to different LLMs.

Table 17: Perplexity and correlation results of our mixed bit allocation.

Methods	Bits	OPT			LLaMA			Gemma
		125M	1.3B	13B	2-7B	2-13B	3-8B	2B
Correlation (τ)		0.910	0.905	0.915	0.931	0.929	0.897	-
Ours (Lat.)	2	712.55	40.28	14.76	9.58	12.57	34.40	16.52 (3 bits)
Ours (Mixed)	2.2	435.84	27.37	13.79	8.97	8.16	29.72	13.89

In addition, we want to clarify that, for each model, we search for the optimal bit allocation with negligible overhead (e.g., 1% 10% of the reparameterization time). For example, it takes 0.5 seconds for searching versus 72 seconds for reparameterizing OPT-125M with a single bit configuration, and 1 minute for searching versus 13 minutes for reparameterizing OPT-13B with a single bit configuration. This is achieved by leveraging the proposed proxy criteria (as shown in Sec. 4.3), instead of searching according to the reparameterization errors, which is time-consuming and requires running models at

each bit. Using the proxy criteria, the bit allocation candidate rankings are highly correlated with the rankings obtained using actual reparameterization errors, with a Kendall τ of 0.910/0.905/0.915 for OPT-125M/1.3B/13B and 0.931/0.929/0.897 for LLaMA-7B/13B/8B, respectively.

G 4-Bit Results and Explanation for Using Lower Bit Widths

We further provide the 4-bit results in Tab. 18. These results show that ShiftAddLLM consistently outperforms the baselines at 4 bits, achieving average perplexity reductions of 0.90/1.32/1.00 and 0.44/0.22/0.02 as compared to OPTQ/LUT-GEMM/AWQ, using OPT models and LLaMA models, respectively.

Table 18: Perplexity comparisons of the OPT models and LLaMA models with 4-bit quantization on WikiText-2. We set the group size as the length of rows for OPT models and 128 for LLaMA models following baselines for fair comparisons.

Method	Bits	OPT							LLaMA			
		125M	350M	1.3B	2.7B	6.7B	13B	30B	1-7B	2-7B	2-13B	3-8B
OPTQ [18]	4	31.12	24.24	15.47	12.87	11.39	10.31	9.63	6.22	5.69	4.98	7.63
LUT-GEMM [48]	4	31.93	24.09	16.15	13.34	12.09	10.40	9.99	5.94	5.78	5.06	6.85
AWQ [38]	4	31.66	7.4e3 (outlier)	15.22	13.19	11.23	-	-	5.78	5.60	4.97	-
Ours (Acc.)	4	**28.72**	**21.59**	**14.98**	**12.65**	**10.95**	**10.20**	**9.63**	**5.76**	**5.58**	**4.96**	**6.46**

We previously considered lower-bit quantization because we aim to push the accuracy-efficiency boundary to lower bits with minimal accuracy compromise. This is meaningful for large-scale LLMs, where even at 3 bits, they remain memory-bound. As analyzed using the Roofline model shown in Figure 5 of [73], for Nvidia A6000 GPUs, the turning point from memory-bound to compute-bound is 200 arithmetic intensity (OPs/bytes). For LLaMA-7B models, all the operators in the decode/generation phase have around or less than 1 arithmetic intensity, as shown in Table 1 of [73]. Even at 4 bits, the arithmetic intensity is approximately $1 \div 3 \times 32 = 8$ (same ops but $4/32$ fewer memory accesses), which is far less than the turning point of 200 and thus remains memory-bound, let alone larger models like LLaMA-70B or beyond. Reducing from 4 bits to 2 bits can help increase the arithmetic intensity and thus the theoretically maximum performance by 2x, from 6144G OPS to 12288G OPS. If memory is not a bottleneck for much smaller cases or prefill stages, higher bits can be used for better accuracy. Our goal is to offer an additional option and trade-off for large, memory-bound cases, without forcing the exclusive use of 2 bits.

H Comparison with MSFP

MSFP [13] is an important prior work that employs a shared exponent across a group of elements and shifts the mantissa accordingly, mimicking multiplication by powers of two. In contrast, we clarify that our approach differs from MSFP in two key aspects:

1. **Nature of Approach**: MSFP uses shared exponents but relies on various shifted mantissa to represent the weights; without this, all weights would collapse to the same value. In contrast, we do not use shared exponents for scaling factors and eliminate the need for mantissa. In particular, each scaling factor is represented as a distinct power-of-two integer (equivalent to the exponents in floating-point numbers, completely removing the mantissa bits). In this way, the multiplication between a floating-point activation and a power-of-two integer scaling factor can be simplified to adding the corresponding integer to the exponent bit of the floating-point activation, as described in Fig. 1 (c). In addition, rather than sharing the exponents, the entire scaling factor in ShiftAddLLM is shared across groups of binary weights in a column/block-wise manner, as illustrated in Fig. 3 (a) and detailed in Sec. 4.2, carefully designed to optimize both weight quantization and output activation errors without conflicts. Hence, there are clear differences between the MSFP datatype and our quantization scheme. In fact, our method is orthogonal to MSFP and can be combined with it by representing input activations in MSFP for more aggressive performance improvements.
2. **Determining Shared Exponents or Scaling Factors**: The method for determining shared exponents in MSFP or shared scaling factors in our quantization scheme is different. MSFP

selects the maximum exponent to share across the bounding-box size, i.e., the number of elements sharing one exponent [13], which is simpler in implementation yet might not be as adaptive. In contrast, in our ShiftAddLLM, the reparameterized binary weights and scaling factors result from multi-objective optimization. This optimization adaptively designs scaling factor patterns to avoid conflicts between optimizing weight errors and optimizing output activation errors.

Finally, in terms of the performance outcomes, MSFP at 4 bits (1-bit sign and 3-bit mantissa) already suffers from large quantization errors, as evidenced by the significant KL divergence shown in Fig. 3 of [13]. In contrast, our ShiftAddLLM at 3 or 4 bits can still achieve comparable accuracy to FP baselines. To directly compare ShiftAddLLM with MSFP, we conducted additional experiments to compare (1) quantization errors and (2) KL divergence using both methods against their floating-point counterparts. We randomly selected ten weight matrices from OPT-350M, quantizing or reparameterizing them using both methods. The results, as summarized in Tab. 19, indicate that ShiftAddLLM consistently outperforms MSFP, achieving lower KL divergence by 0.0065, 0.0271, and 0.0952, and reducing quantization errors by 1707.3, 3251.1, and 5862.0 at 4-bit, 3-bit, and 2-bit quantization, respectively.

Table 19: Comparison between MSFP and ShiftAddLLM with varying bits on KL Divergence and Quantization Error.

Methods	**Bits**	**Avg. KL Divergence**	**Avg. Quant. Error**
MSFP (bounding-box size = 128)	4	0.0117	4129.1
ShiftAddLLM (group size = 128)	4	0.0052	2421.8
MSFP (bounding-box size = 128)	3	0.0434	7859.9
ShiftAddLLM (group size = 128)	3	0.0163	4608.8
MSFP (bounding-box size = 128)	2	0.1485	14355.7
ShiftAddLLM (group size = 128)	2	0.0533	8493.7

I Additional Clarifications on Eyeriss

As emphasized in Sec. 5, our primary focus is on GPU acceleration, specifically through the development of dedicated CUDA kernel support. It is worth noting that, we intentionally did not delve into specific ASIC designs in the main manuscript, which were referenced only to demonstrate potential energy savings.

To clarify the Eyeriss in estimating the energy costs, Eyeriss [8] is a well-known energy-efficient reconfigurable accelerator architecture designed for deep convolutional neural networks (CNNs). It optimizes both dataflow and memory access to reduce energy consumption during neural network processing. In our work, we adapt the Eyeriss architecture by modifying its MAC (Multiply-Accumulate) array, a key component responsible for performing heavy arithmetic computations in CNNs. Instead of using traditional MAC units across the array, we replace selected units with shift, add, and lookup table (LUT) operations, aligning with our proposed ShiftAddLLM approach. This modification significantly reduces both the area and power requirements, with savings ranging from 26% to 89% in different configurations. We refer readers to Fig. 4 of NASA [53], which visually demonstrates the design principles of the overall architecture, and illustrates how replacing traditional MAC units with shift and add operations leads to significant reductions in both area and energy consumption. By adapting these principles, we enhance Eyeriss to better align with the computational needs of both LLMs and ShiftAddLLMs while maintaining power and area efficiency.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of this work in Sec. 5.4.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: This paper does not have theoretical claims.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We clarify the experiment settings and the datasets used in Sec. 5. All LLMs employed in our study are open-source models, aligning with the spirit of reproducibility. In addition, we will release the code and checkpoints upon acceptance to further ensure that others can reproduce our results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: All models and datasets used in this work are open-source. We will release the code and checkpoints upon acceptance to ensure the reproducibility of our results.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide detailed experiment settings, including models, datasets, tasks, evaluation metrics, and baselines, as described in Sec. 5.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Error bars are not reported due to the high computational cost of running all LLM experiments multiple times.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide information on the used GPUs, memory, and latency in Sec. 5.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our research adheres to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the broader societal impact of our research in Sec. 5.4.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our research does not introduce any new data or models; all experiments are conducted on existing data or models. Therefore, this paper does not pose any associated risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, we properly cite all used models, datasets, and related assets.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.