HALO: Hardware-Aware Quantization with Low Critical-Path-Delay Weights for LLM Acceleration

Rohan Juneja[1], **Shivam Aggarwal**[1], **Safeen Huda**[2], **Tulika Mitra**[1], **Li-Shiuan Peh**[1]
[1]National University of Singapore,
[2]OpenAI
{rohan, shivam, tulika, peh}@comp.nus.edu.sg, saf@openai.com

Abstract

Quantization is critical for efficiently deploying large language models (LLMs). Yet conventional methods remain hardware-agnostic, limited to bit-width constraints, and do not account for intrinsic circuit characteristics such as the timing behaviors and energy profiles of Multiply-Accumulate (MAC) units. This disconnect from circuit-level behavior limits the ability to exploit available timing margins and energy-saving opportunities, reducing the overall efficiency of deployment on modern accelerators. To address these limitations, we propose *HALO*, a versatile framework for Hardware-Aware Post-Training Quantization (PTQ). Unlike traditional methods, *HALO* explicitly incorporates detailed hardware characteristics, including critical-path timing and power consumption, into its quantization approach. *HALO* strategically selects weights with low critical-path-delays enabling higher operational frequencies and dynamic frequency scaling without disrupting the architecture's dataflow. Remarkably, *HALO* achieves these improvements with only a few dynamic voltage and frequency scaling (DVFS) adjustments, ensuring simplicity and practicality in deployment. Additionally, by reducing switching activity within the MAC units, *HALO* effectively lowers energy consumption. Evaluations on accelerators such as Tensor Processing Units (TPUs) and Graphics Processing Units (GPUs) demonstrate that *HALO* significantly enhances inference efficiency, achieving average performance improvements of 270% and energy savings of 51% over baseline quantization methods, all with minimal impact on accuracy.

Code — https://github.com/ecolab-nus/HALO

1 Introduction

Transformer-based large language models (LLMs) have grown exponentially, increasing 100-fold every two years, far outpacing the 3.1× improvements in hardware. This widening gap has made inference increasingly costly, as seen with models like LLaMA (65 billion parameters) and GPT-4 (1.76 trillion parameters), which require vast computational resources. Quantization, which reduces model size and cost by lowering bit-width, is crucial for efficiency. However, implementing quantization is challenging because of the diverse, fragmented landscape of hardware accelerators, each optimized for specific quantization techniques and data types. Existing quantization techniques fail to account for these diverse hardware, highlighting the need for adaptable strategies that can optimize model efficiency across different accelerator architectures by leveraging circuit-level characteristics.

Most existing quantization techniques (Aggarwal et al. 2024) overly focus on reducing bit-width without considering hardware-specific factors, often treating critical components like Multiply-Accumulate (MAC) units as a black box. In ML accelerators like Google's TPUs, MAC units used for matrix multiplications central to LLM inference, occupy 77-80% of the chip area and account for 50-89% of total power consumption (Elbtity, Chandarana, and Zand 2024). **Our key observation is that MAC units exhibit significant variability in performance and power characteristics, depending on specific weight patterns after quantization.**

Critical-path-delay in a hardware circuit refers to the time taken by the longest chain of logic operations, which determines the maximum speed at which the overall system can operate. In digital circuits, the path taken by a computation depends on the weight values being processed. Through detailed circuit analysis, we discover that certain weight configurations enable shorter critical-paths, allowing the MAC units to operate at higher frequencies, whereas others result in longer critical-paths that constrain the system's overall operating speed, as illustrated in Fig.4 and Fig.5. This variability highlights an unexplored opportunity to enhance inference performance and energy efficiency through more sophisticated, hardware-aware weight quantization strategies. Yet, current techniques remain oblivious to this crucial quantization-hardware interplay.

Hardware accelerators for LLMs, such as GPUs, use *Dynamic Voltage and Frequency Scaling (DVFS)* to balance performance and power. NVIDIA GA100 GPUs, for instance, support up to 181 DVFS configurations (Ali et al. 2023). Quantization directly influences the critical-path-delays in MAC units, which in turn determines how well DVFS can be utilized to optimize both performance and energy efficiency. Our key insight is that by carefully selecting quantization levels that correspond to favorable critical-path-delays, we can enable higher operating frequencies while maintaining model accuracy.

We present *HALO*, Hardware-Aware LOw critical-path delay quantization framework, illustrated in Fig. 1, which

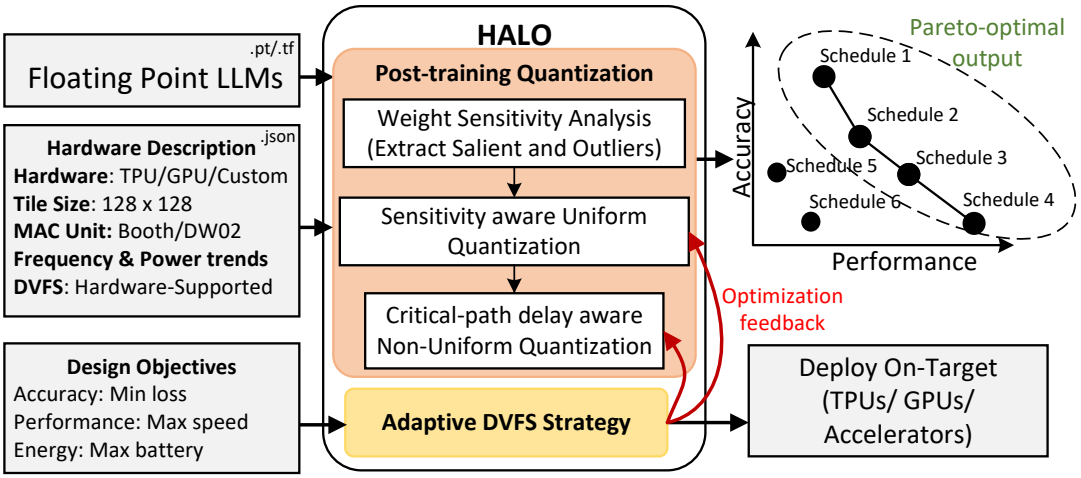


Figure 1: HALO quantization framework, using architectural details to yield Pareto-optimal trade-offs for diverse deployments.

unifies post-training quantization and DVFS into a single hardware-driven optimization strategy. At a high level, HALO first groups weights by their critical-path-delay and then assigns each group the best voltage-frequency setting to speed up "fast" tiles and save energy on "slow" ones. HALO accepts detailed hardware descriptions, such as MAC unit frequency and power profiles, supported DVFS configurations, and tile sizes, along with user-defined design goals for accuracy, performance, and energy efficiency. It outputs a set of Pareto-optimal quantized models, each paired with a corresponding DVFS schedule tailored for efficient deployment on target accelerators such as GPUs and TPUs.

Below, we outline our main contributions.

- We perform comprehensive evaluation of MAC circuit timing and energy variations, identifying weights that are critical for preserving model accuracy through gradient-based sensitivity profiling.
- We introduce HALO, a hardware-aware quantization method utilizing timing, energy, and weight sensitivity to boost LLM efficiency.
- We propose efficient integration techniques for HALO into existing hardware architecture such as TPUs and GPUs, enhancing performance while preserving accuracy.

Our integrated approach delivers significant benefits for LLM inference, achieving on average 270% performance gains and 51% energy savings. These improvements are driven by HALO's ability to co-optimize quantization levels with DVFS operating points, effectively bridging the gap between model compression and hardware adaptation.

2 Related Works

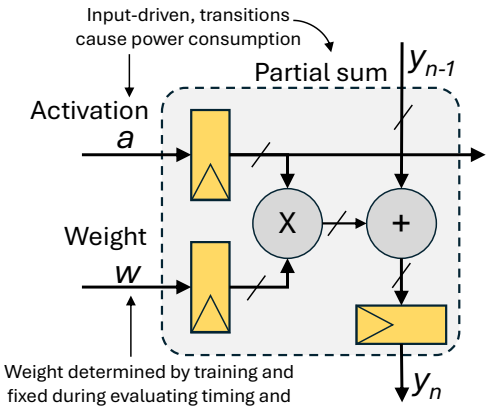
Quantization Methods. Post-training quantization (PTQ) methods such as GPTQ (Frantar et al. 2023) and AWQ (Lin et al. 2024) aim to reduce LLM weights to 3 or 4 bits while preserving key precision. Other approaches like outlier-aware quantization (Lee et al. 2024) and SqueezeLLM (Kim et al. 2024) differentiate between sensitive and non-sensitive weights to guide quantization. SmoothQuant (Xiao et al. 2023) shifts quantization complexity to weights, enabling both weights and activations to be quantized to 8 bits. However, these methods operate at the algorithmic level and do not account for underlying hardware. HALO is the first framework to integrate circuit-level insights, considering timing and DVFS constraints, into the quantization process to improve efficiency without compromising LLM accuracy.

Quantization Accelerators. GOBO (Zadeh et al. 2020) and OlAccel (Park, Kim, and Yoo 2018) use mixed precision to preserve accuracy, while BitFusion (Sharma et al. 2018) adapts precision based on workload needs. GOBO depends on full-precision compute units, and OlAccel encodes outliers using coordinate lists, both requiring custom accelerators. HALO, in contrast, is an orthogonal software-level technique that enhances efficiency without demanding hardware changes, making it suitable for deployment on current AI infrastructure (such as GPUs and TPUs), as well as future accelerators.

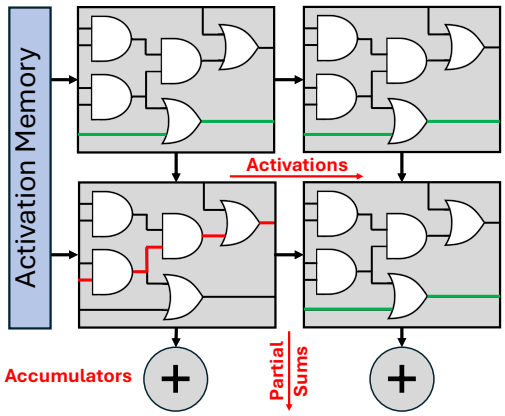
3 Background

The Multiply-Accumulate (MAC) unit is a fundamental component in AI accelerators, playing a significant role in both power consumption and area utilization. As illustrated in Fig. 2a, each MAC unit features three input ports and two output ports. The unit operates by multiplying the weight w with the activation a to produce the product wa. This product is then added to the third input y_{n-1}, resulting in the updated partial sum y_n.

The timing characteristics of a MAC unit are heavily influenced by the specific weight values, as they affect the worst-case critical-path-delays, ultimately constraining the operating frequency. Using Synopsys tools (Inc. 2024), we perform static timing analysis on the 8-bit MAC unit commonly deployed in modern TPUs and GPUs. Fig. 3 illustrates the timing profile for two quantized weights, 64 and -127, with the x-axis representing delay and the y-axis showing the frequency of this delay across all activation transitions. The weight value 64 achieves an operating clock frequency



(a) Overview of the MAC unit, showing the key components.



(b) Bottleneck in a systolic array caused by the slowest MAC unit, limiting clock frequency.

Figure 2: Impact of MAC unit on systolic array efficiency.

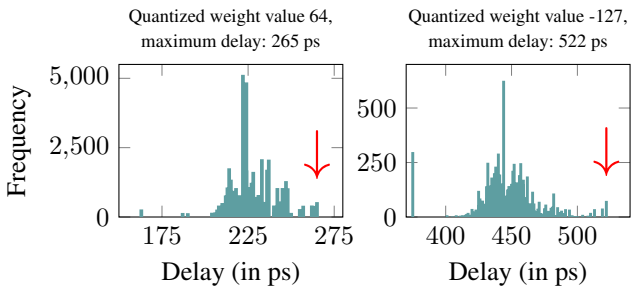


Figure 3: Delay profiles for two weight values. Arrows indicate the maximum delay for each weight across all activations.

of 3.7 GHz, while -127 is limited to 1.9 GHz. Certain bit patterns reduce the number of active signal paths, shortening critical-paths and resulting in faster processing for specific weight values.

Fig. 4 shows the achievable operating frequency, based on

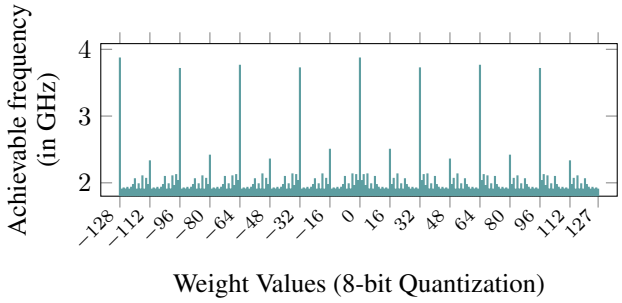


Figure 4: Achievable frequency (GHz) for 8-bit quantized weight values from -128 to 127. Peaks indicate weights with lower critical-path-delays, allowing for higher operating frequencies.

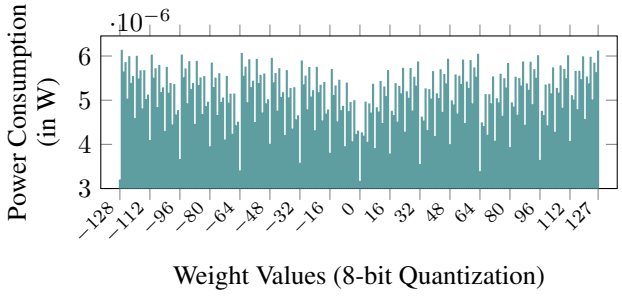


Figure 5: Power consumption (in Watts) for 8-bit quantized weight values ranging from -128 to 127, where lower values reflect decreased power usage due to reduced switching activity.

the maximum delay for each weight value across activation transitions, while Fig. 5 depicts the corresponding power consumption for an 8-bit integer MAC unit. Active power consumption is determined by switching activity, which fluctuates with different weight values. Notably, weights associated with shorter critical-path-delays exhibit lower power consumption, offering potential energy savings. This observed correlation between timing and power characteristics reveals opportunities for optimizing both frequency and energy by strategically selecting weight values for model inference.

The behavior of individual MAC units is crucial in systolic arrays, where performance relies on synchronized operation driven by a global clock, ensuring seamless dataflow across all units. Each MAC unit operates in lockstep to ensure seamless data flow at every clock cycle. This global synchronization, necessary for maintaining data alignment, makes the slowest unit a performance bottleneck. Fig. 2b illustrates the MAC units arranged in a systolic array, emphasizing how the critical-path of the slowest unit restricts the clock frequency. To address this limitation, HALO quantizes weights while accounting for the frequency and energy trends of MAC units and subsequently detects the optimal DVFS configuration. HALO restricts weights within each tile to the same or higher frequency class, enabling execution at the highest safe operating point without violating critical-path timing. All MAC

Algorithm 1: Quantization Framework

Require: calibration dataset X, pre-trained weight matrix W, gradient G
Require: tile size t, quantile threshold k, target frequencies f_1, f_2
Ensure: Quantized weight matrix W_q
1: $W_s, S \leftarrow \text{ExtractSalientValues}(W, G)$ ▷ Isolate values with high saliency
2: $W_o, O \leftarrow \text{ExtractOutliers}(W_s)$ ▷ Separate outlier weights
3: $W^q_{s,o} \leftarrow \text{Quantize}(W_s + W_o)$ ▷ Quantize outliers and salient weights
4: $W_t \leftarrow \text{ReshapeIntoTiles}(\text{PadMatrix}(W_o, t), t)$ ▷ Tile reshaping
5: $\Lambda_{T_k} \leftarrow \text{CalculateTileSensitivities}(G)$ ▷ Compute sensitivity for each tile
6: $M_l, M_h \leftarrow \text{CreateMasks}(M, \text{ComputeAdaptiveK}(\Lambda_{T_k}, k))$ ▷ Classify tiles as low or high sensitivity
7: $W_{l,i}, W_{h,i} \leftarrow W_{t,i} \odot M_{l,i}, W_{t,i} \odot M_{h,i}$ ▷ Apply masks
8: $W_{l,i} \leftarrow \text{Quantize}(W_{l,i}, f_1)$ ▷ Quantize low-sensitivity tiles
9: $W_{h,i} \leftarrow \text{Quantize}(W_{h,i}, f_2)$ ▷ Quantize high-sensitivity tiles
10: $W_q \leftarrow W_{l,i}, W_{h,i}, W_{s,o}$
11: **return** W_q

units within a tile are clocked uniformly, ensuring deterministic behavior and preserving the synchronous dataflow of the architecture. This integrated approach enhances both performance and energy, providing substantial advancement over traditional methods.

4 Quantization Framework

Fig. 6 illustrates HALO's quantization strategy using a simplified 3×3 tile. Traditional quantization (Fig. 6a) applies uniform 8-bit precision to all weights, ignoring MAC-level timing and power variation, which restricts the use of voltage and frequency scaling. In contrast, HALO preserves high precision for salient and outlier weights, which are extracted and encoded as a sparse matrix. The remaining weights are quantized using a critical-path-delay aware scheme and organized into tiles based on their MAC timing characteristics (Fig. 6b). This sparse matrix is reduced to matrix-vector form for efficient execution on dedicated SpMV units. All MAC units within a tile operate at the same frequency, and tiles assigned to the same frequency class are executed together, requiring only a single DVFS transition per class. This design preserves architectural dataflow while minimizing runtime control overhead.

Building on this design, our quantization framework introduces a timing-aware method for LLM inference, detailed in Algorithm 1. It prioritizes weights with low critical-path-delays to reduce latency while maintaining model fidelity. The adaptive method operates across quantization levels and layer sensitivities, optimizing performance by focusing on weights most critical to overall efficiency. The framework consists of three key stages: (1) sensitivity-aware uniform quantization to identify and retain critical weights *(Lines 1–3)*, (2) critical-path-delay aware non-uniform quantization to optimize remaining weights for hardware efficiency *(Lines 4–10)*, and (3) adaptive DVFS, which assigns optimal frequency levels based on tile timing characteristics to maximize performance and energy efficiency.

Stage 1: Sensitivity-Aware Uniform Quantization

The framework begins with a sensitivity analysis of model weights, identifying weights values that can tolerate quantization without significantly affecting accuracy, as outlined in Algorithm 1. The framework initially separates *outliers* (outside the blue lines) and *salient weights* (in red) from normal values, as shown in Fig. 7.

Outliers & Salient Weights : We incorporate outlier removal to manage extreme weight values based on interquartile range scaling. To compute outliers in the weight distribution, we employ the 3σ rule (Guo et al. 2023). Outliers are identified as values lying beyond three standard deviations from the mean.

From the normal values obtained after this distribution, we rely on Taylor series expansion to estimate the most salient weights in the model. Following (Kim et al. 2024), we use an approximation to the Hessian H based on the Fisher information matrix F, which can be calculated over a sample dataset D as

$$F = \frac{1}{|D|} \sum_{d \in D} g_d g_d^\top, \tag{1}$$

where g is the gradient and $H \approx F$. This only requires computing the gradient for a set of samples. For each weight tensor W, the weight sensitivity is computed as $\Lambda_W = F$. Weights with higher Λ_W values are considered more salient due to their significant impact on the model's output. We preserve the top 0.05% of the weights based on this criterion. Cumulatively, both outliers and extremely salient weight values correspond to less than 0.5% of the total weight values. For this reason, we handle these weight values separately and apply per-channel quantization for this set of weight values, isolating them to maintain model precision.

Stage 2: Critical-Path-Delay Aware Non-Uniform Quantization

We leverage non-uniform quantization to more efficiently map the distribution of weights to specific values that reduce the critical-path-delays, thereby optimizing frequency and energy consumption.

Tile-Based Sensitivity Analysis: To optimize the model for efficient inference on hardware, the weight tensors are divided into fixed-size tiles (128×128 by default). Specifically, the sensitivity of each tile is evaluated as the sum of the absolute values of the gradients for each tile, normalized by the size of the tile, based on Eq. 1. For a given kth tile T_k, we compute a *per-tile sensitivity score* Λ_{T_k} using a diagonal approximation of the Fisher information matrix:

$$\Lambda_{T_k} = \frac{\sum_{i,j} g^2_{k,i,j}}{\text{tile\_rows} \times \text{tile\_cols}} \tag{2}$$

where $g_{k,i,j}$ denotes the gradient of the loss with respect to each weight in the k-th tile, and tile_rows $\times$ tile_cols represents the total number of elements within the tile. This score captures the average Fisher information across all weights in the tile, providing a quantitative measure of the tile's sensitivity in relation to its influence on the model's output.

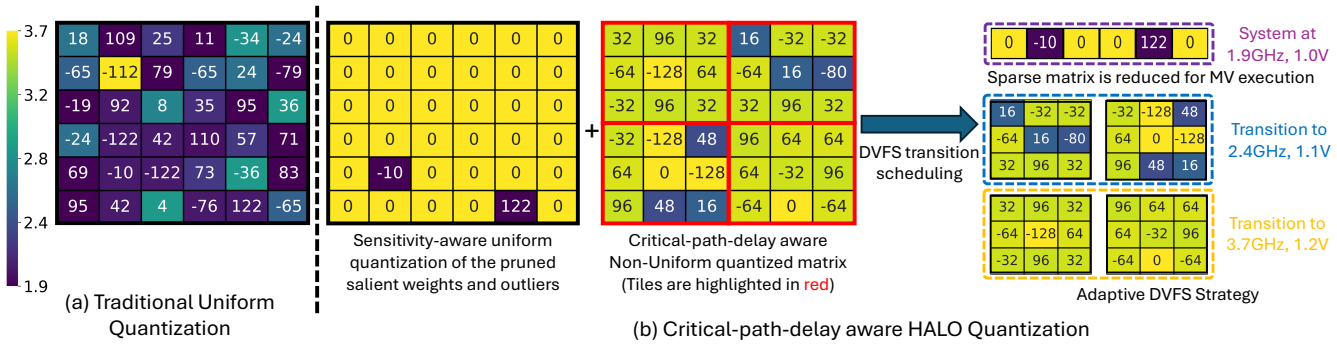


Figure 6: Illustrative example comparing traditional and HALO quantization for 3x3 tile size, highlighting HALO's awareness of critical-path-delay. Only weight quantization is shown; activations are uniformly quantized to 8 bits in both cases. Tile distribution across frequency classes is identical here but depends on user-defined goals for accuracy, performance, and energy.

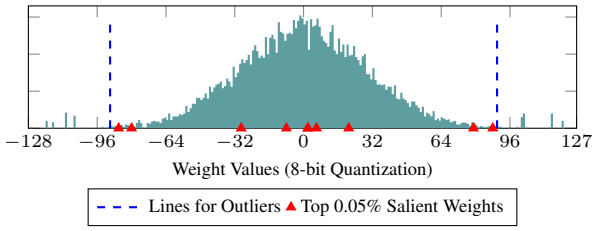


Figure 7: Distribution highlighting sensitive weights important for accuracy.

Tile Sensitivity Mapping : To balance hardware efficiency and model accuracy, tiles in each layer are classified as low-sensitive or high-sensitive based on their relative importance. Determining a fixed sensitivity threshold for each layer is challenging, as weight distributions vary significantly across layers. To address this, we employ a dynamic tile sensitivity mapping strategy that adapts to the cumulative sensitivity distribution of each layer.

The process starts by computing the sensitivity of all tiles in a given layer, derived as the normalized sum of absolute gradient magnitudes within each tile. Sensitivities are then sorted in descending order to rank tiles by importance. A cumulative sum of these sorted sensitivities is then normalized against the total layer sensitivity, generating a cumulative distribution curve from 0 to 1.

The mapping threshold k is derived from this curve and represents the fraction of tiles classified as low-sensitive, ensuring a specified percentage of total sensitivity (e.g., 95%) is retained. Tiles contributing most to overall sensitivity are marked high-sensitive, while the rest are classified as low-sensitive. Mathematically, k is the ratio of the index where cumulative sensitivity exceeds the threshold to the total number of tiles, defaulting to 1.0 if no such index exists.

Once k is determined, boolean masks separate tiles into low- and high-sensitivity categories. Low-sensitivity tiles are quantized more aggressively, while high-sensitivity tiles retain higher precision to preserve performance. The adaptive quantization and computation flow based on the DVFS characteristics are described next.

Stage 3: Adaptive DVFS Strategy

DVFS for Outliers and Salient features:

- Packaging of Salient and Outlier Weights: Salient and outlier weights, exhibiting extreme sparsity, are packaged for efficient computation using a Sparse Matrix-Vector Multiplication (SpMV) engine. The hypersparse weight matrix is compactly stored with a value vector val for non-zero elements and an index vector idx for column positions, reducing memory usage and accelerating computations. The matrix-vector multiplication $A \times b$ for a dense vector $b \in \mathbb{R}^k$ is performed as:

$$\text{res}[i] = \text{val}[i] \times b[\text{idx}[i]], \quad \text{for } i = 0, 1, \ldots, m-1,$$

  where res is the result vector, efficiently leveraging the matrix's sparsity.
- DVFS Configuration Selection: These uniformly quantized hypersparse weights span the entire 8-bit range, necessitating DVFS settings that respect the critical-path-delay for all possible weight values. Guided by MAC characteristic trends, the optimal voltage-frequency (V, f) point is selected to minimize energy consumption while ensuring timing constraints are met:

$$(V, f) = \arg \min_{(V_i, f_i)} E(V_i, f_i)$$
$$\text{given } 1/f_i \geq \text{Critical-Path-Delay}$$

  This approach ensures efficient system performance without sacrificing model fidelity.

DVFS for High- and Low-Sensitivity Weights: For non-uniformly quantized weights on the systolic array, weight tensors are divided into 128×128 tiles, with DVFS settings determined by the tile sensitivity mapping strategy. This tile size aligns with the architecture of modern systolic arrays in TPUs and is critical for balancing the tradeoff between accuracy and performance, optimizing energy efficiency while preserving precision.

The assumed DVFS levels for both GPUs and systolic arrays are summarized in Table 1. We base our DVFS levels on practical hardware parameters: for GPUs, using NVIDIA's publicly available maximum clock frequency of 2.8 GHz (Hardware 2023), and for systolic arrays (similar to TPUs), deriving levels from MAC characteristics.

Hardware	DVFS Levels (Voltage, Frequency)
GPU	(0.9 V, 1.5 GHz), (1.0 V, 2.0 GHz), (1.1 V, 2.8 GHz)
Systolic Array (TPU)	(1.0 V, 1.9 GHz), (1.1 V, 2.4 GHz), (1.2 V, 3.7 GHz)

Table 1: Assumed DVFS levels for GPUs and Systolic Arrays.

- High-Sensitivity Tiles: These tiles are composed of critical weights that are crucial for maintaining model accuracy. As described in the background section, the MAC unit handles 16 high-sensitivity weights, operating at a frequency of 2.4 GHz. These tiles are exclusively made up of these 16 values to ensure precision. The execution is performed at specific DVFS settings (V_i,f_i), where $1/f_i \geq$ the critical delay associated with these weights. While this configuration may lead to higher energy consumption, it significantly boosts tile performance by overclocking the accelerator's global clock to meet the precise timing requirements of these weights.
- Low-Sensitivity Tiles: These tiles are subjected to aggressive overclocking to maximize performance. They contain only 9 weights, each capable of operating at frequencies up to 3.7 GHz. The DVFS settings are fine-tuned to respect the critical-path-delay only for these 9 weights. Despite the high overclocking, the energy increase remains incremental, as switching occurs primarily along the shortest paths in the circuit.

DVFS Transition Scheduling and Overhead Minimization: HALO reduces DVFS overhead by grouping tiles that share the same frequency to run together. Each frequency level is activated once per group, limiting the number of voltage and clock transitions to just a few, each lasting tens of nanoseconds (Bharadwaj et al. 2024) to a few microseconds, significantly shorter than typical LLM inference times (Llama2-13B requires 53 ms, and OPT-30B exceeds 120 ms). Since most models use only two or three frequency levels, the total number of transitions remains low, even in large-scale deployments.

This grouping is a scheduling optimization. Quantization and frequency assignments are determined offline, and execution order does not affect dataflow, timing, or accuracy. Clocking remains uniform within each tile, ensuring that the functional behavior and numerical correctness of the model are preserved.

By leveraging MAC characteristics, this approach achieves an efficient trade-off between processing speed and energy consumption, optimizing the use of the accelerator's resources.

5 Evaluation

This section evaluates the accuracy of LLM models with HALO quantization and demonstrates its speedup and energy efficiency compared to systolic arrays and GPUs.

Implementation Details

Datasets and Models. We evaluate the effectiveness of HALO using various models, including LLaMA2 (Touvron et al. 2023) and OPT (Zhang et al. 2022) family of models. We conduct language modeling evaluation using the C4 (Raffel et al. 2019) and WikiText2 (Merity et al. 2016) datasets. We report *perplexity* as the measure of the performance of the model.

Hyperparameters & Baselines. We evaluate our approach against state-of-the-art quantization methods, including Round-To-Nearest (RTN) quantization, SmoothQuant (Xiao et al. 2023), GPTQ (Frantar et al. 2023), and ZeroQuant (Yao et al. 2022, 2024), under varying weight precision while keeping activations fixed at 8 bits. Quantization is applied to computationally intensive operators such as attention and linear layers, with per-token static quantization used for activations. We evaluate HALO using tile sizes ranging from 128×128 to 32×32 to study tradeoffs between performance and model fidelity.

Hardware Setup. To evaluate the systolic array's performance, we develop a custom simulator and implement the design in SystemVerilog with support for global DVFS control. The design is synthesized using 22nm technology and verified through behavioral simulation. For GPU results, we extend AccelSim (Khairy et al. 2020) to model the NVIDIA 2080 Ti with DVFS settings from Table 1, and estimate energy using AccelWattch (Kandiah et al. 2021) and GPUWattch (Leng et al. 2013).

Accuracy Results

We evaluate HALO against FP16 and WxA8 integer quantization schemes, where activations are quantized to 8 bits and weights to 8, 4, or 3 bits. We compare HALO to RTN, which loses accuracy at lower precisions, and to SmoothQuant, which shifts sensitivity from activations to weights. As shown in Table 2, HALO closely matches FP16 accuracy, with perplexity degradation under 0.5 for most models using the *acc-opt* and *bal* variants. While RTN-W8A8 and SmoothQuant-W8A8 slightly outperform HALO in a few cases, their performance drops sharply at lower bit widths. We include W3A8 and W4A8 baselines as they align with the quantization levels used by HALO. MAC unit delay characteristics allow HALO to group weights into sets of 9 and 16 values, making these baselines relevant for comparison. **Advanced schemes like GPTQ, ZQ-Local, and ZQ-Global perform well on smaller models but fail to match the accuracy of HALO's *acc-opt* and show increasing perplexity with model size, where HALO remains stable.**

To compare heterogeneous quantization schemes, we compute effective bit-width as the weighted average across layers: $B_{\text{eff}} = \sum_i P_i b_i$, with P_i denoting the fraction of parameters at b_i bits. Across different tile sizes, HALO maintains accuracy under aggressive quantization. **For both LLaMA2 and OPT models, HALO preserves performance better as models scale, highlighting the advantages of sensitivity-aware quantization and fine-grained tile selection.** This consistency underscores HALO's suitability for large-scale deployments without compromising numerical robustness.

Systolic Array Performance and Energy

Fig. 8 shows that HALO significantly reduces latency across all models, outperforming FP16 and uniform quantization

PPL↓		WikiText				C4			
		Llama2-7B	Llama2-13B	OPT-1.3B	OPT-30B	Llama2-7B	Llama2-13B	OPT-1.3B	OPT-30B
Ideal	FP16	5.47	4.95	14.72	9.56	7.52	6.98	16.96	11.84
RTN WxA8	x = 8	5.58	4.94	15.23	9.89	7.69	7.04	17.65	12.35
	x = 4	7.36	5.47	81.23	3717.12	10.23	7.71	76.94	4162.58
	x = 3	19480.51	2552.73	13477.29	9122.24	NaN	1951.28	6719.15	6136.87
SmoothQuant (Xiao et al. 2023)	x = 8	5.51	4.93	14.76	9.61	7.56	7.04	16.52	12.03
	x = 4	7.26	5.64	18.47	14.44	10.16	8.00	20.35	31.10
	x = 3	8406.97	572.53	3780.82	1074.64	NaN	1145.67	1188.41	1116.33
GPTQ WxA8 (Frantar et al. 2023)*	x = 4	–	–	15.75	12.79	–	–	15.93	24.14
ZQ-Local WxA8 (Yao et al. 2022)*	x = 4	–	–	15.98	11.69	–	–	16.20	18.96
ZQ-Global WxA8 (Yao et al. 2022)*=	x = 4	–	–	15.77	11.80	–	–	15.83	13.41
HALO (Tile=128)	Perf-opt (BW)	6.37 (3.03)	5.47 (3.03)	16.92 (3.08)	9.95 (3.04)	8.87	7.78	18.43	12.24
	Acc-opt (BW)	5.94 (3.88)	5.20 (3.80)	15.59 (3.96)	9.71 (3.75)	8.23	7.40	17.29	12.05
	Bal (BW)	6.01 (3.75)	5.44 (3.06)	16.06 (3.82)	9.84 (3.42)	8.34	7.73	17.68	12.13
HALO (Tile=64)	Bal (BW)	5.89 (3.62)	5.31 (3.05)	15.82 (3.64)	9.76 (3.40)	8.31	7.50	17.44	12.10
HALO (Tile=32)	Bal (BW)	5.63 (3.33)	5.01 (3.03)	15.56 (3.50)	9.62 (3.24)	8.04	7.19	17.08	12.05

*The results for these techniques are taken from this related work (Yao et al. 2024), and are reported only for models where results are available.

Table 2: LLM accuracy on WikiText (Merity et al. 2016) and C4 (Raffel et al. 2019), measured by perplexity with a sequence length of 2048. Lower values indicate higher accuracy. For HALO, approximate weight bit-width values are reported in brackets alongside perplexity.

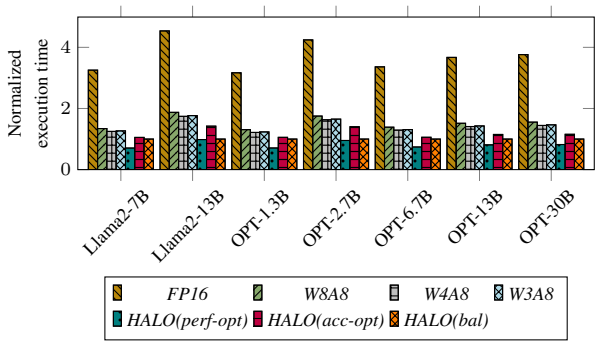


Figure 8: Normalized execution time across quantization methods. Lower values denote faster execution, emphasizing HALO's efficiency.

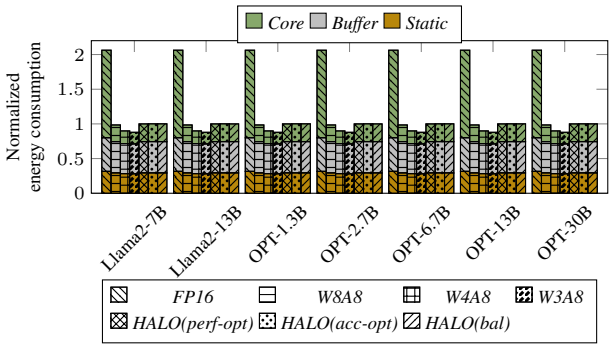


Figure 10: Normalized energy consumed across quantization methods.

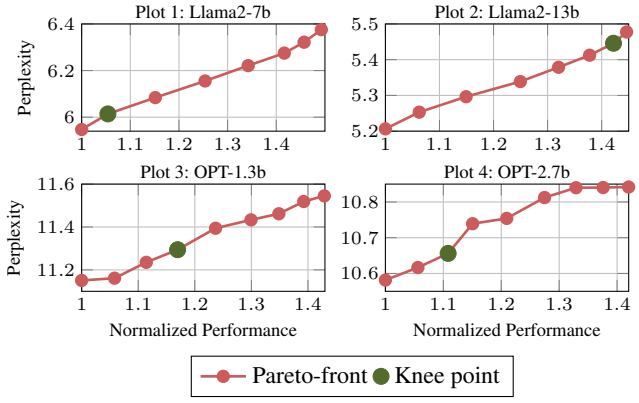


Figure 9: Normalized performance vs. perplexity, showing the knee point for optimal efficiency-accuracy trade-off.

schemes like W8A8, W4A8, and W3A8, which struggle with quantization-sensitive weights. HALO achieves 353% speedup over FP16 and up to 87% over other quantized baselines, while preserving higher accuracy than W4A8 and W3A8. These gains are consistent across different model sizes, highlighting HALO's robustness under aggressive quantization. Execution time for handling outliers and salient weights remains below 1% of total inference time, confirming that tile-level strategies drive most of the improvement. Fig. 9 illustrates the tradeoff between performance and perplexity, with the *bal* configuration corresponding to the knee point, representing an optimal balance for LLM workloads.

Fig. 10 shows energy consumption split into static and dynamic components across core, buffer, and memory. FP16 is the most energy-intensive due to high-precision operations, while W8A8 saves some energy but lacks adaptability to model sensitivity. W4A8 scales better but shows an increase in energy with model size; W3A8 consumes the least energy, although with reduced accuracy. HALO improves energy efficiency by applying voltage and frequency scaling at tile granularity, guided by quantization sensitivity. Its energy use stays within 12% of W3A8 and 10% of W4A8, while offering much faster execution. **Even when operating at a**

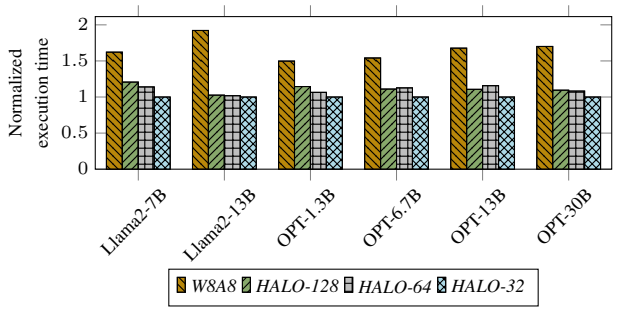


Figure 11: Normalized execution time of the systolic array for different tile sizes. HALO results are shown for the *bal* configuration.

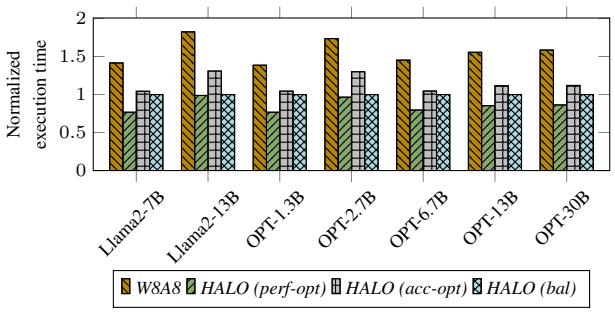


Figure 12: Normalized execution time on GPUs.

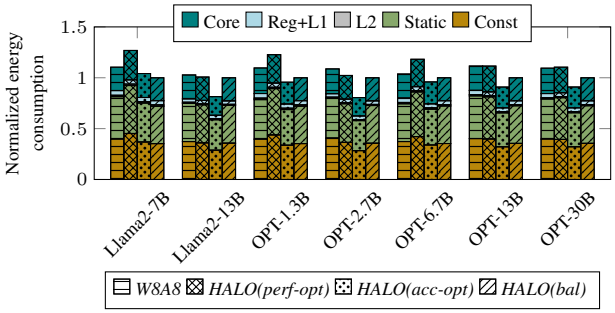


Figure 13: Normalized energy on GPUs.

higher voltage-frequency (VF) point, HALO does not exhibit any rise in total energy compared to W8A8. This is because the increased dynamic power from higher frequency is offset by reduced switching activity within the MAC units, resulting in balanced energy consumption. In systolic arrays, HALO clusters tiles with similar MAC delays, allowing synchronized frequency selection without slower tiles limiting throughput. **Since all tiles in a frequency group run together, the DVFS transitionAPE(sCBoverhead is negligible and amortized across the group.** This lets HALO approach peak performance while preserving timing safety and energy proportionality.

Impact of Tile Size on System Efficiency

Fig. 11 shows HALO's performance across varying tile sizes: HALO-128, HALO-64, and HALO-32, corresponding to 128×128, 64×64, and 32×32 tile dimensions. Smaller tiles, especially 32×32, improve performance by up to 15% over 128×128 and 7% over 64×64. This improvement stems from HALO's ability to quantize a greater number of smaller tiles into higher frequency classes, thereby exploiting finer-grained control over timing and energy characteristics. Smaller tiles also enable the quantization engine to localize aggressive optimization decisions without being constrained by worst-case critical-paths across larger regions.

Table 2 shows that smaller tiles also lead to better perplexity in large models, confirming that finer granularity improves both hardware efficiency and model accuracy. **By reducing synchronization overhead, smaller tiles let each MAC unit run closer to its ideal timing. These results highlight the importance of architecture-aware tile sizing for fully leveraging HALO's quantization and DVFS strategy.** These results emphasize the need for architecture-aware tile sizing strategies to fully exploit the benefits of HALO's quantization and DVFS integration.

GPU Performance and Energy

As shown in Fig. 12, HALO consistently outperforms the *W8A8* baseline on GPU by effectively managing weight sensitivity and aligning quantization levels with hardware capabilities. By selectively assigning frequency classes and applying dynamic voltage and frequency scaling, HALO reduces execution time across all evaluated models. Gains are especially notable in larger models, where quantization-induced bottlenecks are more severe. **As model sizes continue to grow, memory bandwidth and register pressure increase, making hardware-aware quantization and localized DVFS critical for maintaining throughput.**

Fig. 13 shows normalized energy consumption across constant, static, and dynamic sources. Constant power includes peripheral subsystems, while dynamic power comes from DRAM, caches, register files, and compute units. Although *W8A8* uses the least energy due to uniformly low precision, its inability to adapt to workload sensitivity results in limited performance improvements. HALO*(acc-opt)* improves execution time with a slight energy increase, while HALO*(perf-opt)* and HALO*(bal)* prioritize speed and still reduce energy on average, even though tiles run at higher VF points.

HALO improves energy proportionality in the memory and core subsystems by targeting high-frequency execution only where needed, avoiding broad frequency increases that typically lead to energy spikes across the entire device. This highlights HALO's ability to balance performance and energy efficiency through hardware-aware optimization strategies tailored to specific deployment goals.

6 Conclusion

We present HALO, a timing-aware, weight-sensitive quantization framework that improves LLM inference efficiency. By selecting and quantizing weights based on circuit-level insights, HALO reduces power and critical-path-delays, enabling efficient DVFS while remaining hardware-compatible.

Acknowledgments

We thank the reviewers for their feedback and interesting discussion of this work. This research is supported by the National Research Foundation, Singapore, under its Competitive Research Program Award NRF-CRP23-2019-0003 and the Ministry of Education, Singapore, under Tier 3 grant MOE-MOET32024-0003.

References

Aggarwal, S.; Damsgaard, H. J.; Pappalardo, A.; Franco, G.; Preuser, T. B.; Blott, M.; and Mitra, T. 2024. Shedding the Bits: Pushing the Boundaries of Quantization with Minifloats on FPGAs . In *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, 297–303. Los Alamitos, CA, USA: IEEE Computer Society.

Ali, G.; Side, M.; Bhalachandra, S.; Wright, N. J.; and Chen, Y. 2023. Performance-Aware Energy-Efficient GPU Frequency Selection using DNN-based Models. In *Proceedings of the 52nd International Conference on Parallel Processing*, ICPP '23, 433–442. New York, NY, USA: Association for Computing Machinery. ISBN 9798400708435.

Bharadwaj, S.; Das, S.; Mazumdar, K.; Beckmann, B. M.; and Kosonocky, S. 2024. Predict; Don't React for Enabling Efficient Fine-Grain DVFS in GPUs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, ASPLOS '23, 253–267. New York, NY, USA: Association for Computing Machinery. ISBN 9798400703942.

Elbtity, M.; Chandarana, P.; and Zand, R. 2024. Flex-TPU: A Flexible TPU with Runtime Reconfigurable Dataflow Architecture. arXiv:2407.08700.

Frantar, E.; Ashkboos, S.; Hoefler, T.; and Alistarh, D. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv:2210.17323.

Guo, C.; Tang, J.; Hu, W.; Leng, J.; Zhang, C.; Yang, F.; Liu, Y.; Guo, M.; and Zhu, Y. 2023. OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23. New York, NY, USA: Association for Computing Machinery. ISBN 9798400700958.

Hardware, T. 2023. NVIDIA RTX 4070 Allegedly Boosts to 2800 MHz. https://www.tomshardware.com/news/nvidia-rtx-4070-allegedly-boosts-to-2800-mhz. Accessed: 2024-11-20.

Inc., S. 2024. *PrimeTime User Guide*. Accessed: 2024-11-21.

Kandiah, V.; Peverelle, S.; Khairy, M.; Pan, J.; Manjunath, A.; Rogers, T. G.; Aamodt, T. M.; and Hardavellas, N. 2021. AccelWattch: A Power Modeling Framework for Modern GPUs. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, 738–753. New York, NY, USA: Association for Computing Machinery. ISBN 9781450385572.

Khairy, M.; Shen, Z.; Aamodt, T. M.; and Rogers, T. G. 2020. Accel-sim: an extensible simulation framework for validated GPU modeling. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ISCA '20, 473–486. IEEE Press. ISBN 9781728146614.

Kim, S.; Hooper, C.; Gholami, A.; Dong, Z.; Li, X.; Shen, S.; Mahoney, M. W.; and Keutzer, K. 2024. SqueezeLLM: Dense-and-Sparse Quantization. arXiv:2306.07629.

Lee, C.; Jin, J.; Kim, T.; Kim, H.; and Park, E. 2024. OWQ: Outlier-Aware Weight Quantization for Efficient Fine-Tuning and Inference of Large Language Models. arXiv:2306.02272.

Leng, J.; Hetherington, T.; ElTantawy, A.; Gilani, S.; Kim, N. S.; Aamodt, T. M.; and Reddi, V. J. 2013. GPUWattch: enabling energy optimizations in GPGPUs. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, 487–498. New York, NY, USA: Association for Computing Machinery. ISBN 9781450320795.

Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.-M.; Wang, W.-C.; Xiao, G.; Dang, X.; Gan, C.; and Han, S. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proceedings of Machine Learning and Systems*, 6: 87–100.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843.

Park, E.; Kim, D.; and Yoo, S. 2018. Energy-Efficient Neural Network Accelerator Based on Outlier-Aware Low-Precision Computation. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 688–698.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv e-prints*.

Sharma, H.; Park, J.; Suda, N.; Lai, L.; Chau, B.; Kim, J. K.; Chandra, V.; and Esmaeilzadeh, H. 2018. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 764–775.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; Bikel, D.; Blecher, L.; Ferrer, C. C.; Chen, M.; Cucurull, G.; Esiobu, D.; Fernandes, J.; Fu, J.; Fu, W.; Fuller, B.; Gao, C.; Goswami, V.; Goyal, N.; Hartshorn, A.; Hosseini, S.; Hou, R.; Inan, H.; Kardas, M.; Kerkez, V.; Khabsa, M.; Kloumann, I.; Korenev, A.; Koura, P. S.; Lachaux, M.-A.; Lavril, T.; Lee, J.; Liskovich, D.; Lu, Y.; Mao, Y.; Martinet, X.; Mihaylov, T.; Mishra, P.; Molybog, I.; Nie, Y.; Poulton, A.; Reizenstein, J.; Rungta, R.; Saladi, K.; Schelten, A.; Silva, R.; Smith, E. M.; Subramanian, R.; Tan, X. E.; Tang, B.; Taylor, R.; Williams, A.; Kuan, J. X.; Xu, P.; Yan, Z.; Zarov, I.; Zhang, Y.; Fan, A.; Kambadur, M.; Narang, S.; Rodriguez, A.; Stojnic, R.; Edunov, S.; and Scialom, T. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Xiao, G.; Lin, J.; Seznec, M.; Wu, H.; Demouth, J.; and Han, S. 2023. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning*.

Yao, Z.; Aminabadi, R. Y.; Zhang, M.; Wu, X.; Li, C.; and He, Y. 2022. ZeroQuant: efficient and affordable post-training quantization for large-scale transformers. In *Proceedings of the 36th International Conference on Neural Information*

Processing Systems, NIPS '22. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713871088.

Yao, Z.; Wu, X.; Li, C.; Youn, S.; and He, Y. 2024. Exploring post-training quantization in LLMs from comprehensive study to low rank compensation. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press. ISBN 978-1-57735-887-9.

Zadeh, A. H.; Edo, I.; Awad, O. M.; and Moshovos, A. 2020. GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 811–824.

Zhang, S.; Roller, S.; Goyal, N.; Artetxe, M.; Chen, M.; Chen, S.; Dewan, C.; Diab, M.; Li, X.; Lin, X. V.; Mihaylov, T.; Ott, M.; Shleifer, S.; Shuster, K.; Simig, D.; Koura, P. S.; Sridhar, A.; Wang, T.; and Zettlemoyer, L. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068.