

BOLD-Q: Blockwise Outlier-aware Logarithmic Dual-Bias Quantization for Hardware-Efficient LLM Inference

Sungsoo Han, Dahun Choi, and Hyun Kim

Department of Electrical and Information Engineering and Research Center for Electrical and Information Technology
Seoul National University of Science and Technology, Seoul 01811, Korea
hans9805@seoultech.ac.kr, dahun926@seoultech.ac.kr, hyunkim@seoultech.ac.kr

Abstract—Large language models (LLMs) deliver strong natural language processing performance, but ever-growing parameter counts strain memory and power budgets for on-device deployments. Quantization alleviates these costs; however, the outlier-heavy statistics of LLM activations and weights force calibration-based static schemes to retain high-precision fallbacks for dynamically varying values, yielding heterogeneous execution paths and overheads. Microscaling (MX) applies blockwise dynamic quantization with a per-block shared exponent, achieving a homogeneous execution path. Nevertheless, at 4-bit precision, prior work faces three limitations: (i) fixed bins fail to capture block-specific distributions and outliers; (ii) quantization error due to the limited resolution of shared-exponent scaling; and (iii) a lack of co-design approaches that balance model quality and hardware efficiency. We propose *BOLD-Q*, an HW/SW co-design quantization framework that combines the logarithmic number system (LNS) with MX. *BOLD-Q* introduces blockwise Dual-Bias—selected statically for weights via candidate search and dynamically computed for activations—to shift and refine per-block quantization bins, while LNS-based scaling improves distributional fit. On LLaMA-2 7B, *BOLD-Q* limits perplexity increase to +0.32 (W4/A8) and +0.60 (W4/A4), outperforming same-precision baselines. We further design an LNS-MAC systolic array with a lightweight preprocessing row that derives and broadcasts Dual-Bias, eliminating per-PE bias units; within the array, multiplies become log-domain additions, and rescaling is adder-based. Compared with a baseline, *BOLD-Q* reduces area by up to 34.0% and energy by 21.4%, enabling a homogeneous, on-device-friendly, low-precision execution path for LLMs. The code is available at <https://github.com/IDSL-SeoulTech/BOLD-Q>.

Index Terms—Large language models, Quantization, Microscaling, Logarithmic Number System, Systolic arrays

I. INTRODUCTION

Recent advances in high-performance hardware and training infrastructure have accelerated the proliferation of large language models (LLMs), delivering strong performance across diverse natural language processing tasks [1]–[10]. However, these gains have largely been achieved by scaling model size, driving memory and compute requirements to grow

This research was partly supported by Artificial intelligence industrial convergence cluster development project funded by the Ministry of Science and ICT(MSIT, Korea)&Gwangju Metropolitan City and K-CHIPS (Korea Collaborative & High-tech Initiative for Prospective Semiconductor Research) (RS-2025-02305531) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea). The EDA tool was supported by the IC Design Education Center(IDEC), Korea. (Corresponding author: Hyun Kim)

exponentially [11]–[13]. This, in turn, imposes severe resource and power constraints for on-device AI—motivated by privacy preservation, cloud-device workload sharing, and personalization—when deploying LLMs [14]–[20].

To mitigate these constraints, various LLM compression techniques have been explored; among which quantization is a principal approach that jointly reduces memory footprint and computational cost [21]–[33]. However, due to the outlier-prone statistics of LLMs, quantization can incur substantially more projection error than conventional deep neural networks, leading to notable performance degradation. To address this, both calibration-based static quantization and data-format-based dynamic quantization have been proposed. Within the former (*i.e.*, calibration-based static), (i) weight-only methods (GPTQ [24], AWQ [25]) reduce memory via error-compensated quantization and importance-aware outlier preservation, and (ii) weight-activation methods (QServe [28], ATOM [29], QuaRot [30]) mitigate outlier effects through smoothing, outlier isolation, and rotation-based quantization. Nevertheless, both families often retain high-precision floating-point (FP) fallbacks for tasks such as attention score computation and outlier handling, leading to heterogeneous execution and non-trivial hardware overhead.

Accordingly, data-format-based dynamic quantization has gravitated toward Microscaling (MX)—a blockwise data representation with a per-block shared exponent—improving tolerance to outliers [31]–[33]. MX, standardized by the Open Compute Project (OCP) [34], partitions weight/activation tensors into small blocks and normalizes each block’s dynamic range via a shared exponent. Unlike static schemes, MX can quantize activations, such as attention scores, without calibration, thereby enabling a homogeneous execution path across the model. Based on MX, AMXFP [33] and Block-Dialect [32] mitigate low-precision degradation via asymmetric FP scaling and codebook-based assignments, respectively. Nevertheless, outlier-induced errors persist; moreover, asymmetric scaling introduces additional logic along the multiply-accumulate (MAC) and (de)quantization paths, while codebooks impose memory/control overheads, as well as indirect indexing, which reduces arithmetic efficiency. In practice, these designs still rely on multiplication-heavy datapaths, which undermines hardware efficiency.

Despite extensive prior works on MX quantization to enable on-device deployment of LLMs, three key limitations persist.

Limitation 1 (L1): Low-precision fixed quantization bins lack the resolution needed to capture block-level distributional variation and outliers.

Limitation 2 (L2): The coarse step size of per-block shared-exponent scaling degrades robustness to quantization error.

Limitation 3 (L3): A principled HW/SW co-design tailored to block-level quantization remains lacking, limiting the ability to jointly achieve model quality and hardware efficiency.

To address these challenges, we propose *BOLD-Q*, an HW/SW co-design quantization framework that combines the logarithmic number system (LNS)—an expressive yet hardware-friendly numeric representation—with MX. *BOLD-Q* augments MX’s blockwise dynamics with LNS-based Dual-Bias and overcomes the aforementioned limitations as follows: (i) uses Dual-Bias to adaptively reshape bins and accommodate blocks skewed by outliers (\rightarrow L1); (ii) applies finer-grained LNS scaling than a shared exponent to better match per-block distributions (\rightarrow L2); and (iii) deploys an adder-based LNS–MAC systolic array (SA) that replaces multiplies with adds, improving compute and memory efficiency (\rightarrow L3). Our contributions are summarized as follows:

- We propose *BOLD-Q*, an HW/SW co-design quantization framework that combines LNS with MX, providing outlier robustness with high hardware efficiency.
- On the algorithmic side, applying LNS scaling and Dual-Bias yields strong model performance: on LLaMA-2 7B [4] under W4/A8 and W4/A4 quantization, the perplexity (PPL) rises by only 0.32 and 0.60 over the baseline, respectively, outperforming prior works.
- On the hardware side, we implement an LNS-MAC SA tailored to *BOLD-Q* and synthesize this design at the register-transfer level (RTL), achieving up to 34.0% area and 21.4% energy reductions relative to the baseline, thereby demonstrating hardware efficiency.

II. BACKGROUND AND RELATED WORKS

A. Microscaling Quantization

The MX format [34], a blockwise dynamic representation, splits activation/weight tensors into small blocks and computes a per-block maximum exponent. Elements are aligned and scaled to that exponent, then rounded to the target format. Each block, therefore, uses a single shared exponent with low-precision codes per element. Weights can be quantized statically (offline), while activations and the attention path are quantized dynamically at run-time on a per-block basis. This small-block scaling localizes outliers, reducing error in the MXFP8 regime while lowering calibration requirements. However, at 4-bit precision, the presence of a large outlier within a block can inflate the shared exponent, thereby severely reducing the effective resolution of the remaining elements.

NxFP [31] introduces a per-block nano-mantissa and adaptive microexponent to absorb outliers, but still relies on FP operations and leaves activation quantization unresolved.

BlockDialect [32] performs codebook-based quantization by assigning, for each block, the code that minimizes the quantization error. However, codebook storage and indirect indexing incur memory/control overhead, and a coarse 0.5-step size limits granularity. AMXFP [33] applies FP-based asymmetric scaling to weights and activations, improving low-precision quality over MX, but it adds logic across accumulation and (de)quantization to handle asymmetry, and FP scaling remains substantially costlier than shared-exponent schemes.

B. Logarithmic Number System

MX-based methods primarily mitigate degradation via per-block adaptation to outliers. Leveraging the LNS property, which transforms multiplication into addition, is advantageous for attenuating outlier effects in a hardware-efficient manner. In LNS, a real number x is represented by a sign bit s and its logarithmic magnitude $y = \log_2|x|$, stored as the pair (s, y) :

$$x = (-1)^s 2^y. \quad (1)$$

where y is treated as a fixed-point quantity composed of integer and fractional bits. Accordingly, in the LNS domain, multiplication reduces to addition as follows:

$$P = \log_2(a \times b) = \log_2 a + \log_2 b = y_a + y_b. \quad (2)$$

where a, b are positive real numbers and $s_p = s_a \oplus s_b$. This identity allows multiplication-centric computations to be recast as additions in the log-domain, significantly reducing the cost of the multiplier. By contrast, expressing linear-domain addition in the log-domain requires a correction term as:

$$\log_2(a + b) = \alpha + \log_2(1 + 2^{\beta-\alpha}). \quad (3)$$

where $\alpha = \max\{\log_2 a, \log_2 b\}$ and $\beta = \min\{\log_2 a, \log_2 b\}$. Evaluating the correction term $\log_2(1 + 2^{\beta-\alpha})$ typically requires *lin-to-log/log-to-lin* transformations. In practice, these transformations are approximated using lookup tables (LUTs) or Mitchell’s approximation [35]; however, both introduce non-trivial hardware overhead due to table storage and access, as well as the associated approximation logic.

Accordingly, prior work [36] trims conversion cost by applying a power-of-two (PoT) approximation after log-domain quantization. HLQ [36] offsets log-mapping bias via train-time shifts of quantization points and replaces multiplications with PoT-based shift-add, cutting resource use. However, performing the associated *log-to-lin/lin-to-log* transformations at every accumulation step incurs nontrivial hardware overhead. LNS-LLM [37] instead builds an LNS-based LLM accelerator with LNS-tailored processing elements (PEs) and LUT-driven *log-to-lin/lin-to-log* conversion, and mitigates activation outliers by selectively allocating extra bits—yielding strong HW/SW efficiency. Nevertheless, it leaves the attention path unaddressed, thereby necessitating additional heterogeneous execution paths.

LNS offers an expressive numeric representation and replaces multiplies with adds, promising a hardware-efficient design. However, prior LNS work either fails to realize an efficient LNS compute system or does not support full-model

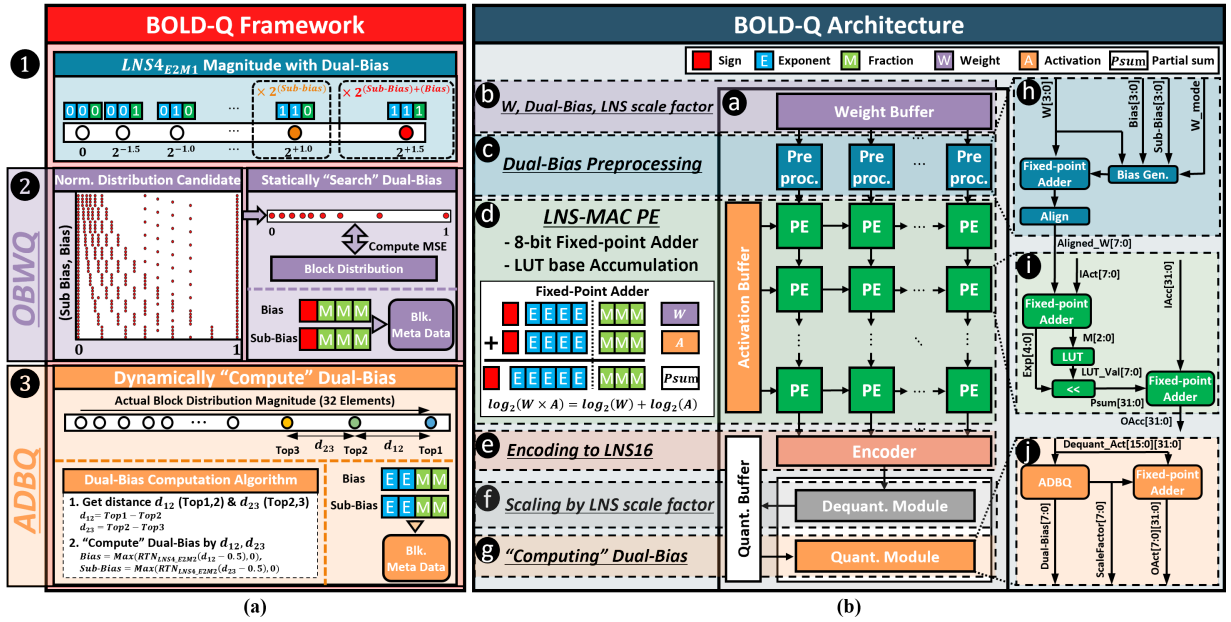


Fig. 1: Overview of BOLD-Q: (a) Overall framework and (b) dedicated architecture.

quantization, thereby requiring heterogeneous execution paths. In contrast, MX provides a full-model homogeneous path but suffers notable degradation in model performance at low-precision. Although subsequent MX-based methods introduce blockwise adaptation to mitigate outlier effects, many overlook hardware considerations and thus incur significant resource and control overheads. These observations motivate a combined approach: for on-device deployment of full-model, low-precision quantization, it is necessary to couple LNS and MX.

III. PROPOSED METHOD: BOLD-Q

Fig. 1 provides a comprehensive overview of the proposed *BOLD-Q* quantization framework (Fig. 1(a)) and the corresponding hardware architecture (Fig. 1(b)). Section III details the framework, and Section IV describes the architecture.

A. Overall Framework

Fig. 1(a) illustrates (i) the notion of Dual-Bias (1), (ii) its offline search for weights (2), and (iii) its runtime computation for activations (3). In 1, Dual-Bias is injected into $LNS4_{E2M1}$ data—a 4-bit LNS format with {sign:1, exponent:2, fractional:1} bits, where M denotes the fractional field. Injecting Dual-Bias into the bins flexibly reshapes the block distribution; the procedure for determining Dual-Bias differs between weights and activations. For weights (2), we perform an offline static search over candidate distributions to select Dual-Bias. For activations (3), we perform a dynamic computation to track instantaneous block statistics and outliers. Unless otherwise noted, these procedures operate on data scaled by an $LNS8_{E5M3}$ scale factor.

B. Dual-Bias Injection into LNS

We introduce Dual-Bias to address the limitation that fixed low-precision quantization bins cannot adequately represent

block-specific distributions. Dual-Bias represents a paired offset composed of an LNS4 Sub-Bias and Bias. As indicated by 1 in Fig. 1(a), Dual-Bias is injected into the top two bin values of $LNS4_{E2M1}$, namely $q_1 = 2^{1.5}$, $q_2 = 2^{1.0}$, and can be expressed as follows:

$$q_1 = 2^{1.5+Sub-Bias+Bias}, \quad q_2 = 2^{1.0+Sub-Bias}. \quad (4)$$

Here, all remaining bins are left unchanged. By selectively biasing q_1 and q_2 , the quantization levels are adjusted flexibly to better match block-specific distributions and attenuate the influence of outliers.

C. Offline Block-Specific Dual-Bias for Weight Quantization

To determine the optimal Dual-Bias for weights, we propose an offline block-specific Dual-Bias for weight quantization (OBWQ). OBWQ performs a static, per-block search for the Dual-Bias that minimizes the mean squared error (MSE). The left panel of 2 in Fig. 1(a) shows representative candidate distributions induced by different (Sub-Bias, Bias) pairs, enabling MSE-optimal fitting across diverse weight blocks. For a given weight block (upper-right panel of 2), OBWQ evaluates the MSE for each candidate distribution; we define MSE as:

$$MSE(\delta_s, \delta_b) = \frac{1}{B} \|W_b - \hat{W}_b(\delta_s, \delta_b)\|_2^2. \quad (5)$$

where δ_s and δ_b denote the Sub-Bias and Bias, respectively; W_b is the original weight vector of block b ; $\hat{W}_b(\delta_s, \delta_b)$ is the dequantized weight vector obtained by applying the Dual-Bias (δ_s, δ_b) together with the block scale; B is the block size (number of elements); and $\|\cdot\|_2$ denotes the Euclidean (L2) norm. We evaluate the MSE over all candidate pairs, select the Dual-Bias that minimizes the MSE, and store it—along with

the block’s scale factor—as metadata (lower-right panel of **2**). Repeating this procedure for every block yields a more accurate representation of block-specific distributions than fixed-bin quantization. Moreover, since weight distributions typically exhibit smaller outliers than activations, we encode the biases in LNS4_{E0M3} .

D. Adaptive Dual-Bias Quantization

Activations and attention exhibit significant input-dependent variability; in particular, softmax over attention logits amplifies quantization errors, making calibration-based static quantization insufficient to preserve model performance. Dynamic quantization is therefore required; however, unlike weights, search-based methods that demand extensive exploration are impractical at runtime. To address this, we propose adaptive Dual-Bias quantization (ADBQ), which leverages Dual-Bias for dynamically varying data. ADBQ extends the blockwise top-1 principle of dynamic MX to compute Dual-Bias dynamically. Concretely, as shown at the top of **3** in Fig. 1(a), we extract the top-1 to top-3 magnitudes from each block and then apply the Dual-Bias computation algorithm depicted in the lower-left panel of **3**. The algorithm computes the distances between the first/second maxima and between the second/third maxima. Whenever a gap is at least 0.5, we convert only the excess beyond 0.5 to an additive bias by rounding it to the nearest value representable in LNS4_{E2M2} , as follows:

$$\text{Bias} = \max(\text{RTN}_{\text{LNS4}_{E2M2}}(d_{12} - 0.5), 0). \quad (6)$$

$$\text{Sub-Bias} = \max(\text{RTN}_{\text{LNS4}_{E2M2}}(d_{23} - 0.5), 0). \quad (7)$$

where d_{12} and d_{23} denote the distances between the top-1/top-2 and top-2/top-3 magnitudes, respectively, and $\text{RTN}_{\text{LNS4}_{E2M2}}(\cdot)$ denotes rounding to the nearest value representable in LNS4_{E2M2} . Because the native step size of LNS4_{E2M1} is 0.5, deviations beyond this threshold distort the distribution; accordingly, we add only nonnegative (additive) biases as a corrective shift. The activation/attention path exhibits larger outliers than the weight path. Because the algorithm applies an additive correction on top of existing bins, we encode the biases in LNS4_{E2M2} to provide a wider representable range than LNS4_{E0M3} —albeit with a coarser step size. Finally, as indicated in the lower-right panel of **3**, the computed Dual-Bias is stored as metadata and used to quantize the subsequent layer.

IV. ARCHITECTURE DESIGN: BOLD-Q ARCHITECTURE

A. Architecture Overview

We also propose a hardware architecture optimized for *BOLD-Q*, summarized in Fig. 1(b). **a** depicts a 32×32 SA of PEs with a weight stationary dataflow, operating at a default precision of W4/A8. In the attention pathway, the key-value (KV) cache is quantized to 4 bits, while the query (Q) and attention scores use 8 bits; Dual-Bias is applied only to the 4-bit paths. Activations stream from left to right across the array, and weights stream from top to bottom. Partial sums accumulate downward along columns: each PE accumulates

its LNS–MAC result into the incoming 32-bit fixed-point accumulated value from above and forwards the updated value to the PE below. At the bottom row, the column-wise accumulation outputs feed the encoder, the dequantization module, and the quantization module. In **b**, the weight buffer stores the per-block weights, along with the corresponding Dual-Bias and LNS scale factor produced by OBWQ. Similarly, the activation buffer stores the per-block activations, Dual-Bias, and LNS scale factor obtained from ADBQ. On the weight path, a dedicated preprocessing row (**c**) performs (i) block-wise alignment and format normalization and (ii) application of the per-block Dual-Bias prior to injection into the PEs. Each PE (**d**) implements an 8-bit fixed-point, adder-based LNS–MAC: multiplications are replaced by additions in the log-domain, and the linear-domain accumulation required for the fraction component is realized via a LUT-based `log-to-lin` approximation, minimizing conversion overhead.

The SA’s column-wise accumulations are passed to the encoder, which converts the 32-bit fixed-point sums to the log-domain (`lin-to-log`) and outputs LNS16-encoded values. In **e**, the proposed encoder realizes the `lin-to-log` operation by enhancing Mitchell’s approximation with a piecewise-linear (PWL) correction and a small LUT for residual compensation. Instead of instantiating an encoder in every PE, we place encoders only at the SA output, reducing the instance count by a factor of 32 (*i.e.*, to 1/32 at the system level). In **f**, the LNS16 stream is dequantized using per-block W/A LNS scale factors. As in the PE stage, these factors are applied via fixed-point operations by using an LNS8_{E5M3} scale factor, resulting in minimal overhead relative to shared-exponent scaling. Finally, in **g**, for each block, we compute the Dual-Bias dynamically via ADBQ and re-quantize with the LNS scale factor to produce the next-layer inputs.

In summary, the proposed architecture implements a homogeneous LNS–MX pipeline consisting of (i) buffer streaming, (ii) Dual-Bias computation and alignment in a preprocessing row, (iii) core compute on an LNS–MAC SA, (iv) LNS16 encoding followed by dequantization, and (v) re-quantization to produce the next-layer inputs. The hardware architecture is designed to support this pipeline effectively, avoiding FP16 fallbacks and relying only on localized `log-lin` conversions.

B. Preprocessing Module and Processing Elements

In **h** of Fig. 1(b), the preprocessing module selects the bias configuration (*e.g.*, LNS4_{E0M3} , LNS4_{E2M2}) according to `W_mode` and generates the corresponding bias based on the `W[2:0]`. It then performs fixed-point additions in the log-domain and pre-aligns the result to `Aligned_W[7:0]` for subsequent operations with LNS8 activations before streaming to the SA. By deploying 32 preprocessing units to offload Dual-Bias handling, the design eliminates the per-PE bias logic that would otherwise be replicated across the full 32×32 array, thereby significantly reducing hardware overhead.

In **i**, each PE implements an LNS–MAC. In the multiplication stage, the PE receives the `IAct[7:0]` from the activation buffer and `Aligned_W[7:0]` from the preprocessing

TABLE I: Quantization Results for Various LLMs on WikiText-2 (perplexity ↓).

Method	Linear (W/A)	Atten. (W/A)	Scale Factor	Block Size	OPT 6.7B	OPT 13B	LLaMA1 7B	LLaMA1 13B	LLaMA2 7B	LLaMA2 13B	LLaMA3 8B	Mistral 7B
Baseline	16/16	16/16	-	-	10.86	10.13	5.68	5.09	5.47	4.88	6.14	5.25
GPTQ [24]	4/16	16/16	FP16	128	10.93	10.17	5.83	5.20	5.63	4.99	6.56	5.39
AWQ [25]	4/16	16/16	FP16	128	10.93	10.21	5.78	5.19	5.60	4.97	6.54	5.37
OmniQuant [26]	4/16	16/16	FP16	128	10.96	10.20	5.77	5.17	5.58	4.95	-	-
AMXFP [33]	4/16	16/16	FP8	32	10.96	10.34	5.84	5.21	5.64	4.98	-	5.39
BOLD-Q (ours)	4/16	16/16	LNS8	32	10.89	10.16	5.76	5.15	5.55	4.95	6.40	5.32
ANT [38]	8/8	16/16	FP16	-	19.72	4E+3	8.52	7.49	8.79	20.52	-	-
QServe [28]	4/8	4/16	FP16	128	-	-	5.89	5.25	5.70	5.08	6.76	5.42
MXFP [34]	4/8	4/8	PoT8	32	12.27	11.64	6.81	5.91	6.75	6.01	-	5.88
AMXFP [33]	4/8	4/8	FP8	32	11.40	10.99	5.99	5.35	5.85	5.20	-	5.48
BOLD-Q (ours)	4/8	4/8	LNS8	32	11.20	10.74	5.97	5.28	5.79	5.14	6.70	5.41
ANT [38]	4/4	16/16	FP16	-	9E+3	4E+4	80.13	96.71	189.72	165.19	-	-
Quarot [30]	4/4	4/16	FP16	Channel	-	-	6.34	5.58	6.10	5.40	8.17	5.80
ATOM [29]	4/4	4/16	FP16	128	-	-	6.16	5.46	6.12	5.31	7.76	5.76
MXFP [34]	4/4	4/4	PoT8	32	22.51	12.88	10.20	7.80	11.18	6.98	11.17	9.43
AMXFP [33]	4/4	4/4	FP8	32	13.06	11.90	6.25	5.52	6.22	5.47	7.72	5.71
BlockDialect [32]	4/4	4/4	PoT8	32	11.63	-	6.39	-	6.33	-	7.87	5.74
BOLD-Q (ours)	4/4	4/4	LNS8	32	11.45	10.97	6.18	5.45	6.07	5.37	7.34	5.56

Entries with unreported or unreproducible results are marked “-”.

row, and realizes LNS multiplication with an 8-bit fixed-point adder. In the accumulation stage, the fractional field $M[2:0]$ of the product is used as the LUT address. The LUT stores $2^{m/8}$ values in Q1.7 (8-bit) fixed-point format, where m denotes the 3-bit fractional field. The `LUT_Val[7:0]` is shifted by `Exp[4:0]` to form `Psum[31:0]`; this term is then accumulated with the incoming `IAcc[31:0]` to produce `OAcc[31:0]`, which is forwarded to the next row of the array. As a result, the `log-to-lin` reduces to a LUT access, and the linear-domain accumulation is performed via shift-add without a dedicated multiplier. Moreover, the `lin-to-log` conversion is performed only at the SA output by the encoder, thereby minimizing conversion overhead.

C. Quantization module

In ① of Fig. 1(b), the quantization (Quant.) module takes `Dequant_Act[15:0][31:0]`—a 32-element block vector in LNS16—and, through ADBQ, produces the per-block `Dual-Bias[7:0]` and `ScaleFactor[7:0]`. For re-quantization, it applies `ScaleFactor[7:0]` to `Dequant_Act[15:0][31:0]` using fixed-point additions and outputs `OAct[7:0][31:0]` in LNS8_{E4M3}. When targeting 8-bit activations, the module excludes Dual-Bias and performs quantization using only the LNS scale factor. The Quant. module reduces hardware cost for handling activation outliers by computing Dual-Bias with a lightweight ADBQ block governed solely by the simple threshold-and-round rules in Eqs. (6)–(7); as in the Dequant. module, re-quantization performs fixed-point scaling, further limiting overhead.

V. EXPERIMENTAL RESULTS

A. Algorithm Performance Evaluation

Experimental Setup. To demonstrate the effectiveness of *BOLD-Q*, we evaluate perplexity (PPL ↓) on the WikiText-2

dataset [39] using the lm-eval-harness framework [40] across five families of LLMs: LLaMA-1 (7B/13B) [3], LLaMA-2 (7B/13B) [4], LLaMA-3 8B [5], OPT (6.7B/13B) [9], and Mistral 7B [2]. To quantify the accumulation error introduced by the 8-bit LUT-based `log-to-lin` approximation in *BOLD-Q*’s LNS-MAC, we conducted experiments with 10,000 test vectors assuming an accumulation length of 32, observing an error of 0.26%. We then reflected this minor error in model inference by injecting randomized perturbations at the output stage, aligned with the block size.

Weight only quantization. Table I presents PPL results for *BOLD-Q* and representative quantization methods. Under weight-only quantization (*i.e.*, linear layers at W4/A16 and attention at W16/A16), *BOLD-Q* yields PPL increases below 0.10 relative to the baseline across all models, delivering near-lossless quality compared with competing research. Calibration-based static methods, such as GPTQ, AWQ, and OmniQuant, offer certain advantages in terms of block size relative to AMXFP and *BOLD-Q*; however, they require FP16 scaling during dequantization, resulting in a mixed-precision execution path. Compared with AMXFP, both approaches replace the MX shared exponent with alternative scale factors; however, by additionally applying Dual-Bias, *BOLD-Q* attains consistently lower PPL. In terms of metadata, AMXFP maintains separate FP8 scales for positive and negative values, adding 16 bits per block. *BOLD-Q* likewise uses an LNS8 scale together with an 8-bit Dual-Bias, totaling 16 bits per block.

Weight & Activation Quantization. With the 8-bit linear activation setting in Table I, QServe attains the lowest PPL on many models, while *BOLD-Q* achieves comparable performance with marginal differences. Unlike *BOLD-Q*, QServe quantizes the KV cache but executes attention at W4/A16,

yielding a heterogeneous execution path. By contrast, *BOLD-Q* runs attention with W4/A8 and attains near state-of-the-art quality among methods that avoid FP16 fallbacks. Notably, on Mistral 7B and LLaMA-3 8B, *BOLD-Q* achieves even lower PPL loss than QServe, indicating robustness despite the difference in attention precision. Furthermore, compared with full-model quantizers such as MXFP and AMXFP, *BOLD-Q* consistently delivers superior PPL across most configurations.

With the 4-bit linear activation setting, ATOM retains FP16 operations along the attention path and thus attains favorable PPL on a few models; however, across most models, *BOLD-Q* achieves the best PPL while operating in a strictly 4-bit configuration. Turning to MX-based methods, naive MXFP exhibits severe degradation at 4-bit owing to shared-exponent scaling and fixed bins. AMXFP and Block-Dialect partially mitigate this degradation but are still clearly outperformed by *BOLD-Q* among strictly 4-bit methods. Overall, these results demonstrate that the full-model quantization framework *BOLD-Q*—combining LNS scaling with OBWQ/ADBQ—consistently preserves strong language modeling quality (low PPL) at low precision.

B. Hardware Performance Evaluation

Hardware Implementation. We implement the proposed SA as an RTL design in Verilog. We synthesize all components in a commercial 28nm technology library using Synopsys Design Compiler with timing constraints set to 500 MHz clock frequency and report area and energy (estimated from post-synthesis power). All architectures target a W4/A8 precision configuration; for ANT, a W4/A8 PE is constructed by composing two W4/A4 PEs. For AMXFP, the PE and the Quant./Dequant. Modules are modified to support asymmetric scale factors.

Hardware Resource Evaluation. Table II presents the component-area breakdown for *BOLD-Q* and the comparison architectures. Figs. 2(a) and 2(b) report architecture-level totals normalized to AMXFP: total area (a) and energy (b). In Fig. 2(a), MXFP, ANT, and *BOLD-Q* reduce the area by 23.2%, 23.7%, and 34.0% relative to AMXFP. As summarized in Table II, these savings stem from (i) eliminating per-PE bias logic by applying Dual-Bias in a preprocessing row and using an LNS-MAC PE, and (ii) collapsing the per-PE Encoders into output-side Encoders at the SA boundary, reducing the number of encoder instances to 1/32. Moreover, the Dequant. module in *BOLD-Q* exhibits an area comparable to MXFP because LNS scaling is realized via addition; by contrast, AMXFP handles asymmetric FP scales on separate positive/negative accumulators and fuses them via FP addition, making its Dequant. area roughly 10× that of *BOLD-Q*. Although the Quant. module of *BOLD-Q* incurs some overhead relative to MXFP to support ADBQ, it still achieves a 56.4% area reduction compared to AMXFP, thereby providing an efficient dynamic quantization path.

Consistent with the area trends, Fig. 2(b) presents normalized energy reductions of 17.6%, 7.5%, and 21.4% for MXFP, ANT, and *BOLD-Q*, respectively, relative to AMXFP.

TABLE II: Component-area breakdown for *BOLD-Q* and baseline architectures.

Arch.	Core		Others		
	Component (μm^2)	Num.	Area (mm^2)	Dequant. (mm^2)	Quant. (mm^2)
AMXFP	PE (521.35)	1024	0.542	0.03	0.039
	Encoder (243.83)	32			
MXFP	PE (443.20)	1024	0.458	0.003	0.008
	Encoder (121.91)	32			
ANT	PE (423.77)	1024	0.432	0.017	0.017
	Decoder (12.64)	32			
BOLD-Q (Ours)	Encoder (121.91)	32	0.383	0.003	0.017
	PE (372.41)	1024			
	Preproc. (89.97)	32			
	Encoder (128.00)	32			

"Preproc." denotes the preprocessing module.

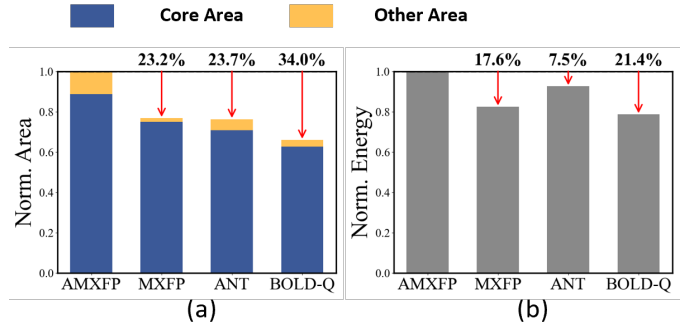


Fig. 2: Comparison of normalized area and energy between *BOLD-Q* and other architectures.

While MXFP must support both normal and subnormal exponent regimes—incurring additional control and datapath logic—only one path is active at runtime, and switching on the inactive path is limited, yielding energy savings that are proportionally larger than area reductions. Overall, *BOLD-Q* delivers the largest reductions in both area and energy among the evaluated architectures, underscoring its suitability for on-device deployment.

VI. CONCLUSION

In this study, we propose *BOLD-Q*, an HW/SW co-design quantization framework that combines LNS with MX. It leverages Dual-Bias via OBWQ/ADBQ to mitigate outliers and preserve model quality. In addition, an adder-based LNS-MAC systolic array—exploiting multiplication-as-addition in the log-domain—achieves higher efficiency and fewer resources than prior designs. In experiments, we observe up to 34.0% area and 21.4% energy reductions relative to the baseline. At the same time, *BOLD-Q* consistently maintains competitive PPL at low precision. Overall, *BOLD-Q* establishes a homogeneous LNS-MX execution path suited to on-device low-precision inference and offers an HW/SW solution with potential to expand the feasibility of deploying high-performance LLMs on resource-constrained devices.

REFERENCES

- [1] T. Le Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé *et al.*, “Bloom: A 176b-parameter open-access multilingual language model,” 2023.
- [2] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, “Mistral 7b,” 2023.
- [3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [4] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” *arXiv preprint arXiv:2307.09288*, 2023.
- [5] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.
- [6] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, “Improving language understanding by generative pre-training,” 2018.
- [7] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [9] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, “Opt: Open pre-trained transformer language models,” *arXiv preprint arXiv:2205.01068*, 2022.
- [10] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocar, M. Debbah, É. Goffinet, D. Hesslow, J. Launay, Q. Malartic *et al.*, “The falcon series of open language models,” *arXiv preprint arXiv:2311.16867*, 2023.
- [11] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, “Palm: Scaling language modeling with pathways,” *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.
- [12] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhume, G. Zerveas, V. Korthikanti, E. Zhang, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro, “Using deepspeed and megatron to train megatron-turing nl3 530b, a large-scale generative language model,” 2022.
- [13] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.
- [14] M. Xu, F. Qian, Q. Mei, K. Huang, and X. Liu, “Deeptype: On-device deep learning for input personalization service with minimal privacy concern,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 4, pp. 1–26, 2018.
- [15] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge intelligence: Paving the last mile of artificial intelligence with edge computing,” *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.
- [16] J. Yuan, C. Yang, D. Cai, S. Wang, X. Yuan, Z. Zhang, X. Li, D. Zhang, H. Mei, X. Jia *et al.*, “Mobile foundation model as firmware,” in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024, pp. 279–295.
- [17] W. Yin, M. Xu, Y. Li, and X. Liu, “Llm as a system service on mobile devices,” *arXiv preprint arXiv:2403.11805*, 2024.
- [18] S. I. Lee, K. Koo, J. H. Lee, G. Lee, S. Jeong, S. O, and H. Kim, “Vision transformer models for mobile/edge devices: a survey,” *Multimedia Systems*, vol. 30, no. 2, p. 109, 2024.
- [19] J. Shin and H. Kim, “L-tta: Lightweight test-time adaptation using a versatile stem layer,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 39 325–39 349, 2024.
- [20] D. Cai, S. Wang, Y. Wu, F. X. Lin, and M. Xu, “Federated few-shot learning for mobile nlp,” in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–17.
- [21] D. Choi and H. Kim, “Gradq-vit: Robust and efficient gradient quantization for vision transformers,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 15, 2025, pp. 16 019–16 027.
- [22] N. J. Kim, J. Lee, and H. Kim, “Hyq: Hardware-friendly post-training quantization for cnn-transformer hybrid networks,” in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24. International Joint Conferences on Artificial Intelligence Organization*, vol. 8, 2024, pp. 4291–4299.
- [23] S. Jeong, S. Lee, and H. Kim, “Lowgradq: Adaptive gradient quantization for low-bit cnn training via kernel density estimation-guided thresholding and hardware-efficient stochastic rounding unit,” in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–2.
- [24] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, “Gptq: Accurate post-training quantization for generative pre-trained transformers,” *arXiv preprint arXiv:2210.17323*, 2022.
- [25] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, “Awq: Activation-aware weight quantization for on-device llm compression and acceleration,” *Proceedings of machine learning and systems*, vol. 6, pp. 87–100, 2024.
- [26] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo, “Omniquant: Omnidirectionally calibrated quantization for large language models,” *arXiv preprint arXiv:2308.13137*, 2023.
- [27] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, “Smoothquant: Accurate and efficient post-training quantization for large language models,” in *International conference on machine learning*. PMLR, 2023, pp. 38 087–38 099.
- [28] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han, “Qserve: W4a8kv4 quantization and system co-design for efficient llm serving,” *arXiv preprint arXiv:2405.04532*, 2024.
- [29] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasicki, “Atom: Low-bit quantization for efficient and accurate llm serving,” *Proceedings of Machine Learning and Systems*, vol. 6, pp. 196–209, 2024.
- [30] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, P. Cameron, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, “Quarot: Outlier-free 4-bit inference in rotated llms,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 100 213–100 240, 2024.
- [31] Y.-C. Lo, G.-Y. Wei, and D. Brooks, “Nanoscaling floating-point (nxfp): Nanomantissa, adaptive microexponents, and code recycling for direct-cast compression of large language models,” *arXiv preprint arXiv:2412.19821*, 2024.
- [32] W. Jang and T. Tambe, “Blockdialect: Block-wise fine-grained mixed format quantization for energy-efficient llm inference,” *arXiv preprint arXiv:2501.01144*, 2025.
- [33] J. Lee, J. Park, J. Kim, Y. Kim, J. Oh, J. Oh, and J. Choi, “Amxfp4: Taming activation outliers with asymmetric microscaling floating-point for 4-bit llm inference,” *arXiv preprint arXiv:2411.09909*, 2024.
- [34] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf *et al.*, “Microscaling data formats for deep learning,” *arXiv preprint arXiv:2310.10537*, 2023.
- [35] J. N. Mitchell, “Computer multiplication and division using binary logarithms,” *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, 1962.
- [36] D. Choi, J. Park, and H. Kim, “Hlq: Hardware-friendly logarithmic quantization aware training for power-efficient low-precision cnn models,” *IEEE Access*, 2024.
- [37] P. Haghi, C. Wu, Z. Azad, Y. Li, A. Gui, Y. Hao, A. Li, and T. T. Geng, “Bridging the gap between llms and lns with dynamic data format and architecture codesign,” in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 1617–1631.
- [38] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, “Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization,” in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.
- [39] S. Merity, C. Xiong, J. Bradbury, and R. Socher, “Pointer sentinel mixture models,” *arXiv preprint arXiv:1609.07843*, 2016.
- [40] L. Gao, J. Tow, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, K. McDonell, N. Muennighoff *et al.*, “A framework for few-shot language model evaluation,” *Version v0. 0.1. Sept*, vol. 10, pp. 8–9, 2021.