Lattice-Based Vector Quantization for Low-Bit Quantization-Aware Training

Rishika Kohli[1], Soma S. Dhavala[1], Shaifu Gupta[1], Manoj Singh Gaur[1]
[1]Indian Institute of Technology, Jammu.

Quantization is an effective approach for deploying deep learning models on resource-constrained hardware, but maintaining accuracy and training stability at extreme low precision remains a major challenge. In this work, we study lattice-based vector quantization (VQ) as a practical alternative to scalar quantization for low-bit quantization-aware training (QAT). We develop a unified quantization pipeline that integrates structured lattice projections into both QAT and post-training quantization (PTQ), supporting multiple lattice choices—including E8 and D4—via a fused projection operator with straight-through estimation.

Through extensive experiments across a wide range of bit-widths, lattice parameterizations, and training regimes, we show that lattice-based VQ consistently enables stable training and meaningful accuracy below 2 bits, where scalar quantization and existing PTQ methods typically underperform or are unavailable. In this low-bit regime, exploiting geometric structure across weight blocks improves robustness by reducing overload and stabilizing optimization, while at moderate and higher bit-widths, performance differences narrow and simpler quantization schemes become sufficient. We further analyze the role of lattice choice, dynamic-range scaling, and overload behavior, and demonstrate that explicit overload control is central to reliable low-bit performance. Finally, we show that lattice-based QAT extends beyond binary classification and weight-only quantization, supporting multi-class tasks, joint weight–activation quantization, and transformer encoders such as BERT, achieving substantial compression with controlled accuracy degradation.

1. Introduction

Accommodating deep learning (DL) models on edge devices is often challenging due to the high computational requirements [1, 2]. To address the gap between current DL models and the requirement of edge deployment, researchers have explored three broad directions [3]: pruning the model, designing lightweight architectures, and employing model quantization. Together, these approaches form the foundation of current strategies for enabling efficient DL on edge devices. In this work, we explore this third direction of model quantization. Model quantization maps high-precision floating-point weights and activations to lower-precision fixed or floating-point representations, offering substantial memory and compute savings.

Recent work highlights that model quantization for low-bit inference can substantially alleviate memory bottlenecks, particularly when combined with fast lookup-table–based computation [4]. This is crucial for edge hardware, where on-chip storage and bandwidth are severely constrained. The field has recently seen a push towards 1-bit [5] quantization, where linear layers are aggressively quantized but can still maintain competitive accuracy. These advances further motivate the exploration of low-bit scalar and vector quantization schemes, as pursued in this work.

Our Contributions. A major portion of inference cost in DL models arises from matrix–matrix or matrix–vector multiplications of the form WX, where W denotes the model's weight matrix and X is the input activation. Quantizing the weights (and activations) applies a mapping $Q : \mathbb{R} \to \mathcal{Q}_b$ where $\mathcal{Q}_b$ denotes the set of fixed/floating-point values representable using b-bit precision. For

Third Conference on Parsimony and Learning (CPAL 2026).

example, common b-bit formats include integer types such as `int8/qint8` and low-bit floating-point types such as `float16` and `bfloat16`, as implemented in PyTorch[1]. In this work, two popular model quantization paradigms: *quantization-aware training* (QAT: training-time quantization) and *post-training quantization* (PTQ: forward-only quantization), are explored. We develop a unified quantization framework that examines both these paradigms for low-bit deployments and can work for multiple lattice types. Within the QAT setting, we study two families of quantizers. In *scalar quantization* (SQ), each weight is treated independently and works well when the data is uniformly spread along each coordinate. Whereas, in *vector quantization* (VQ), small blocks of weights (such as 8-D vectors) are quantized jointly and are more effective when the data has a non-uniform distribution, allowing it to use bits more efficiently and achieve better rate–distortion performance. In this work, we use structured lattices for VQ. Beyond proposing the quantization pipeline, our study systematically examines several practical questions that arise when pushing bit-widths to extreme limits.

- Can QAT reliably perform better than PTQ in the low-bit regime? (*Section 4.1*)
- Does lattice parameterization affect quantized model performance? (*Section 4.2*)
- Can lattice-based VQ be integrated into transformer encoders such as BERT while preserving downstream task performance? (*Section 4.3*)
- Can QAT be introduced without any initial FP32 warm-up phase? (*Appendix*)
- Does reducing overload (residual error between the original and quantized weights exceeding the bit budget) lead to stable performance and mitigate degradation at low precision? (*Appendix*)
- Does the proposed framework generalize to other lattice families beyond E8 (e.g., lower-dimensional lattices)? (*Appendix*)
- Can joint quantization of weights and activations be performed within the proposed framework while maintaining stability at low precision? (*Appendix*)
- Does lattice-based QAT extend beyond binary classification to more general task settings such as multi-class classification? (*Appendix*)

To affirmatively answer these questions in our study, we conduct extensive experiments across multiple datasets and architectures. *Note:* For clarity, quantization precision is classified here as low-bit precision (< 4 bits), mid-bit precision (4–12 bits), and high-bit precision (> 12 bits). Our code is available on `https://github.com/mlsquare/coset`.

2. Related Work

In this work, we examine model quantization in both QAT and PTQ modes.

QAT incorporates quantization into the training process via straight-through estimation, allowing the model to adapt to the noise introduced by low-bit computation, but requires access to both the model parameters and the training data. DoReFa-Net [6] showed that models with low-bit weights and activations can be trained using deterministic quantization, while gradients require stochastic quantization. [7] introduces a practical QAT approach that simulates 8-bit quantization during training using affine (scale + zero-point) quantization, enabling accurate integer-only inference on edge hardware.

PACT [8] proposes a learnable activation-clipping mechanism where the clipping parameter is optimized during training to reduce quantization error, enabling stable low-bit activations and near–full-precision accuracy in CNNs. LSQ [9] improves training-time quantization by learning the quantizer step size directly from task loss, achieving strong results for 2–4 bits weights and activations. LSQ+ [10] extends this framework with learnable offsets and a more robust initialization scheme, enabling effective asymmetric activation quantization and better stability for architectures

[1] https://docs.pytorch.org/docs/stable/tensor_attributes.html

with non-ReLU activations. SAT [11] further examines the instability of low-bit training and introduces scale-adjusted updates alongside calibrated activation-clipping gradients, resulting in more stable optimization and competitive accuracy in the quantized mode.

Brevitas [12] is a QAT framework that implements scalar uniform quantization using STE-based scale learning. It provides a strong and widely adopted baseline for quantization methods. To the best of our knowledge, all existing QAT methods operate purely in the scalar domain, and VQ has not been explored in QAT settings, largely due to the computational overhead with codebook construction and finding the nearest codebook - referred to as Closest Vector Problem (CVP) in the lattice literature.

On the other hand, **PTQ** aims to quantize a trained DL model into a low-precision version without performing any retraining. It requires only the pretrained weights and, optionally, a small unlabeled calibration dataset to estimate activation ranges. Early PTQ methods mainly used simple techniques such as rounding weights to the nearest quantization level, adjusting clipping ranges [13, 14], or applying bias correction [15]. However, these heuristic approaches often suffer from large accuracy drops, especially at 4-bit precision, because they do not consider how weight perturbations affect the model's output or task loss. To mitigate this, [16] introduced a data-driven rounding strategy that learns whether each weight should round up or down, using a layer-wise reconstruction loss to match the FP32 layer output. Building on this, BRECQ [17] showed that block-wise reconstruction optimizing quantization over blocks of consecutive layers rather than each layer independently, further stabilizes error propagation and yields better accuracy.

More recently, QuIP [18] shows that preprocessing weights using structured orthogonal transforms improves incoherence, making weights easier to quantize and significantly reducing PTQ error. Building on this, QuIP#[19] extends this idea with a faster Hadamard-based transform, E8-lattice VQ, and a lightweight fine-tuning stage, achieving good 2–3 bit PTQ performance. Extending this, CALDERA [20] builds on QuIP# by exploiting the approximate low-rank structure of LLM weight matrices to enhance quantization quality.

Existing QAT methods are exclusively scalar because learning multidimensional codebooks and performing lattice projections impose substantial computational and implementation burdens during training. Even though QuIP# demonstrates the benefits of structured orthogonal transforms and E8-lattice VQ, these techniques have not been extended to the QAT setting, nor have they been explored within a hierarchical (multi-digit) lattice decomposition. In particular, QAT for LLMs is relatively underexplored [18]. As a result, the field lacks a unified framework that enables the use of lattice-based hierarchical VQ in QAT. Addressing these gaps is essential for understanding whether the advantages of lattice-based quantization in PTQ can translate to the QAT setup.

3. Methodology

In this section, we organize the quantization work into two complementary approaches: vector and scalar quantization.

3.1. Vector quantization

VQ originates from classical information-theoretic insights that coding vectors rather than scalars achieves strictly better rate distortion performance [21]. It maps multi-dimensional vectors to a finite set of codewords rather than quantizing each element independently, where each codeword represents a representative vector in the quantized space. Given a codebook $\mathcal{C} = \{C[i]\}$, the quantizer assigns an input vector x to the nearest codeword $Q(x) = \arg\min_i \|x - C[i]\|_2$. Classical VQ methods typically learn the codebook from data, whereas lattice-based VQ employs pre-determined, structured codebooks defined by the underlying lattice geometry. In this work, we focus on lattice-based VQ, so we briefly introduce some lattice definitions.

3.1.1. Preliminaries

A lattice $L \subset \mathbb{R}^n$ is a discrete additive subgroup generated by integer linear combinations of linearly independent basis vectors. Formally, $L = \{\sum_{i=1}^{n} z_i b_i \mid z_i \in \mathbb{Z}\}$, where $b_1, b_2, \ldots, b_n$ are linearly independent vectors that form a basis of the lattice. L is closed under vector addition and reflection [22], and its points are uniformly distributed in $\mathbb{R}^n$ according to the chosen basis. A lattice can also be represented compactly using a generator matrix $G \in \mathbb{R}^{n \times n}$ whose columns are the basis vectors, such that $L = \{Gz \mid z \in \mathbb{Z}^n\}$.

For a point $x \in \mathbb{R}^n$, the nearest-neighbor lattice quantizer $Q_L(x)$ maps x to its closest lattice point (Algorithm 3 in [23]), $Q_L(x) = \arg\min_{v \in L} |x - v|$. Each lattice point $v \in L$ acts as a *codeword* in the quantization scheme, and the quantized representation of x is given by the selected codeword.

The Voronoi region $\mathcal{V}_L$ of a lattice is defined as the set of points in $\mathbb{R}^n$ that are closer to the origin than to any other lattice point, $\mathcal{V}_L = \{x \in \mathbb{R}^n : |x| \leq |x - v|,\ \forall v \in L \setminus \{0\}\}$. Each lattice point has a corresponding Voronoi cell obtained by translating $\mathcal{V}_L$ to that point. These Voronoi cells tile the space without overlap and form the fundamental domain for nearest-neighbor quantization.

Two geometric quantities play an important role in quantization: the *packing radius*, defined as the radius of the largest sphere that fits inside a Voronoi cell, and the *covering radius*, defined as the radius of the smallest sphere that completely covers the Voronoi cell [24]. Lattices with large packing radius and small covering radius generally yield better quantization performance.

Several classes of lattices are commonly used in quantization, including the integer lattice $\mathbb{Z}^n$, the D_n lattice (integer vectors with even coordinate sum), and the highly symmetric E_n lattices. The E_8 **lattice** (also used in QuIP#, but in a PTQ setting) is an even, unimodular lattice in $\mathbb{R}^8$, whose points are given by $E_8 = \left\{x \in \mathbb{Z}^8 \cup (\mathbb{Z}^8 + \frac{1}{2}\mathbf{1}) : \sum_{i=1}^{8} x_i \equiv 0 \pmod 2\right\}$ [24]. Its Voronoi cell has the smallest known covering radius among all 8-D lattices, making E_8 particularly well suited for VQ, where inputs are projected to their nearest lattice point (codeword).

3.1.2. Prior works on Lattice quantization

Mukherjee et al. [25] introduced successive refinement in lattice VQ, where vectors are quantized through a hierarchy of Voronoi-shaped lattice codebooks that progressively reduce distortion at each stage. Later, Fuchs [26] proposed embedded Voronoi codes to refine lattice VQ. It combines differently scaled Voronoi codebooks to yield a 1-bit-per-dimension progressive bitstream while retaining the structural advantages of lattice quantization.

Hierarchical Nested Lattice Quantization (HNLQ) [4] extends classical nested-lattice VQ by decomposing a fine-lattice projection into M radix-q stages. Rather than directly representing $x \in \mathbb{R}^d$ using a single large Voronoi codebook, HNLQ expresses the projection $Q_L(x)$ as a sum of hierarchical correction terms $\hat{x} = \sum_{m=0}^{M-1} g_m(x)$, where each digit $g_m(x)$ belongs to a scaled Voronoi codebook $q^m A_q$ with $A_q = L \cap (q\mathcal{V}_L)$. The digits arise from recursive nested projections defined by $g_m(x) = Q_L^{\circ m}(x) - Q_L^{\circ(m+1)}(x)$, where $Q_L^{\circ m}$ denotes m successive applications of the scaled lattice quantizers. By telescoping, $\hat{x} = Q_L(x) - Q_L^{\circ(M)}(x)$, and thus $\hat{x} = Q_L(x)$ whenever the M-fold nested projection vanishes, i.e., $Q_L^{\circ M}(x) = 0$, which corresponds to the no-overload regime. Because each digit is represented using $d \log_2(q)$ bits, the total rate is $dM \log_2(q)$ bits, replacing a single exponentially large codebook with M compact sub-codebooks that admit efficient lookup-table implementations. Here, we state a lemma from [4].

Lemma 1 (Correctness of Hierarchical Reconstruction [4])**.** *HNLQ reconstructs as* $\hat{x} = \sum_{m=0}^{M-1} g_m(x)$*, and* $\hat{x} = Q_L(x) \iff Q_L^{\circ M}(x) = 0$ *Thus, overload occurs precisely when the* M*-fold coarse projection does not vanish.*

When overload does not occur, i.e., $Q_L^{\circ M}(x) = 0$, the telescoping sum reduces to $\hat{x} = Q_L(x)$. In this regime, the hierarchical refinement perfectly reconstructs the nearest lattice point of L. Moreover, because each digit $g_m(x)$ lies in a scaled sub-codebook $q^m A_q$, the overall representation uses

only $dM \log_2(q)$ bits. This structure enables efficient lookup-table–based inner-product computation with table size scaling as $2^{2dR/M}$ (compared to 2^{2dR} for flat vector quantization).

3.1.3. Our approach

A common limitation of traditional VQ approaches is that they rely on arbitrarily learned codebooks, which lack geometric structure and are difficult to integrate uniformly across neural network layers. Our method instead incorporates a fixed, geometry-derived E_8 lattice into MLP and BERT linear layers via the fused HNLQ operator in Algorithm 1, combining (i) dynamic-range normalization, (ii) blockwise E_8 lattice projection, and (iii) an optional STE for QAT.

A key novelty of our work is extending HNLQ to a full QAT regime. Unlike traditional VQ, whose codebook is to be learnt from data, lattice VQ uses a fixed geometric codebook with strong distance properties. This enables a unified, architecture-agnostic quantizer that operates consistently across both PTQ and QAT. However, controlling the dynamic range of the input to operate HNLQ effectively is very critical.

Dynamic-range calibration. Given a weight matrix $W_{\text{original}} \in \mathbb{R}^{O \times I}$, we normalize the dynamic range of each row using a *row-wise heuristic scale* $\beta_{\text{row},i}$. For row i, we estimate its empirical standard deviation σ_i and define $\beta_{\text{row},i} = Y_{\max}/C_b\sigma_i$ where C_b is a safety constant and $Y_{\max} = \Delta_0(q^M - 1)/2$ is a lattice-controlled range parameter. The normalized weights are $\widetilde{\mathbf{W}}_{i,:} = \beta_{\text{row},i}\mathbf{W}_{i,:}$. Derivation of normalization constant, under scalar quantization regime, is present in [27]. In principle, β may be applied per row or per block after tiling into n-D vectors. While block-wise scaling provides finer local control, it requires storing additional scaling factors. We therefore adopt *per-row* scaling in all experiments.

Unlike learnable quantization methods such as LSQ+ [10], which optimize scale and offset parameters via backpropagation, our dynamic-range calibration is entirely *fixed and heuristic*, derived solely from weight statistics and lattice geometry. In addition, we employ an exponential moving-average estimate of feature means for centering, which improves training stability without introducing extra trainable parameters.

Blockwise E_8 quantization. Since the E_8 lattice operates in $\mathbb{R}^8$, each row of $\widetilde{\mathbf{W}}$ is reshaped into 8-D blocks. For a block vector $\widetilde{\mathbf{w}} \in \mathbb{R}^8$, we compute its nearest lattice point (CVP) through the projection operator (detailed in *Appendix*: *Algorithm 3*) $\widehat{\mathbf{W}} = Q_{\text{E8}}(\widetilde{\mathbf{w}})$ which evaluates candidates in both D_8 and its coset $D_8 + (0.5)^8$ with parity correction, guaranteeing that $\widehat{\mathbf{w}}$ lies exactly in Λ_{E_8}. The operator Q_{E8} used in Algorithm 1 can be instantiated either as the *exact* nearest-neighbor (*Appendix Algorithm 3*) or as the *Babai* nearest-plane approximation (*Appendix Algorithm 4*). Exact computes the true closest lattice point, while the Babai method provides a faster but approximate projection. After quantizing all blocks, the matrix is reconstructed as $\widetilde{\mathbf{W}}_{\text{quant}}$ and rescaled; optionally, elementwise clipping is applied for numerical stability during training.

Lattice QAT via a fused STE approximation. Unlike HNLQ, which performs full hierarchical encoding and decoding (Algorithms 1 and 3 of [4]), our method implements *Lemma 1* [4] and collapses the entire lattice quantizer into one fused operator that is sufficient for QAT. During the forward pass, each 8-D block is simply mapped to its nearest E8 lattice point, and during the backward pass, we apply an STE of the form $\tilde{x} = x + \text{SG}(Q_{E8}(x) - x))$, where $\text{SG}(\cdot)$ is the stop-gradient operator. This treats the projection as identity in the gradient path. This fused design removes the need for explicit encoding and residual loops that are only necessary for storage or communication, not for learning. The fused nearest-point operator alone is enough for stable QAT, even at varied q and M, making the approach significantly simpler and more efficient than the original HNLQ pipeline.

Relation to HNLQ and the role of (q, M). Our method is a simpler adaptation of HNLQ where the fine lattice is E_8, and the hierarchical digits are implicitly produced through the fused projection with flexibility and minimal overhead. In HNLQ, the nearest lattice point is expressed through radix q digits across M nested layers. In our QAT setup, (q, M) still determines the effective representable radius $Y_{\max}$ and the trade-off between quantization accuracy and overload risk.

Reconstruction and optional clipping. After all blocks are projected, the normalized matrix is rescaled via $\beta_{\mathrm{row},i}^{-1}$. An optional elementwise clipping stabilizes the early QAT phase by preventing large outlier weights from destabilizing the lattice projection.

Algorithm 1 Fused E8 vector quantization

Input: Weights $W \in \mathbb{R}^{O\times I}$; radix q; digits M; projection Q_{E8}; block size 8; clip τ.
Output: $\widehat{W}$.
for $i = 1..O$ **do**
  $\beta_i = Y_{\max}/(C_b\sigma(W_{i,:}))$, $\widetilde{W}_{i,:} = \beta_i W_{i,:}$
end
Split $\widetilde{W}$ into 8-D blocks $\widetilde{\mathbf{w}}_k$.
for *each block* k **do**
  $\widehat{\mathbf{w}}_k = Q_{\mathrm{E8}}(\widetilde{\mathbf{w}}_k)$
end
Assemble $\widehat{W}$ and invert scaling: $\widehat{W}_{i,:} = \widehat{W}_{i,:}/\beta_i$.
Clip: $\widehat{W} = \mathrm{clip}(\widehat{W}, -\tau, \tau)$.

3.2. Scalar quantization

SQ depicted in Algorithm 2 follows the same normalization approach used in the VQ, but quantizes each weight independently rather than operating on n-D blocks. Here, scaling factor $\beta_i = \frac{2^{b-1}-1}{C_b\,\sigma_i}$ maps the dynamic range of that row into the representable interval of a b-bit signed uniform quantizer. The rest things remain the same. For a unified framework and to make it comparable to VQ, $\widetilde{W}$ is partitioned into contiguous 8-D blocks, but unlike VQ, no inter-component structure is exploited. Each block $\widetilde{\mathbf{w}}_k$ is quantized elementwise using the scalar operator $\widehat{\mathbf{w}}_k = Q_{\mathrm{SQ}}(\widetilde{\mathbf{w}}_k)$ where $Q_{\mathrm{SQ}}(\cdot)$ (detailed in *Appendix Algorithm 5*) is a uniform rounding quantizer.

Although SQ treats each weight independently, it can be viewed as a special case of VQ. In the VQ formulation, quantization is performed by projecting each 8-D block onto the lattice generated by a basis matrix G. When the generator matrix is chosen as the identity, $G = I$, the lattice reduces to $\mathbb{Z}^8$, and the nearest-lattice projection becomes simple element-wise rounding. Thus, SQ corresponds exactly to VQ with an identity lattice, and the two differ only in the structure of the underlying quantizer. Because the weights are still grouped into 8-D blocks for implementation consistency, SQ remains architecturally comparable to VQ. This demonstrates that our QAT framework is generic and extensible: the same normalization, row-wise scaling, and fused STE machinery seamlessly handle both structured lattice-based VQ and unstructured SQ under a unified formulation.

Existing SQ-QAT methods primarily differ in how quantization ranges are determined. Brevitas [12] adopts an empirical approach with fixed design choices, while LSQ+ [10] employs learnable, data-driven scaling parameters optimized during training. In contrast, ACIQ [27] derives clipping thresholds analytically based on assumed weight or activation distributions. Our SQ follows an empirical but adaptive scaling strategy based on simple heuristics without any learnable parameters. Implementation details of integrating SQ into the framework are provided in the *Appendix*.

Algorithm 2 Scalar quantization

Input: Weights W; bit-width b; scalar quantizer Q_{SQ}; block size 8; clip τ.
Output: $\widehat{W}$.
for $i = 1..O$ **do**
  $\beta_i = (2^{b-1}-1)/(C_b\sigma(W_{i,:}))$, $\widetilde{W}_{i,:} = \beta_i W_{i,:}$
end
Split $\widetilde{W}$ into 8-D blocks $\widetilde{\mathbf{w}}_k$.
for *each block* k **do**
  $\widehat{\mathbf{w}}_k = Q_{\mathrm{SQ}}(\widetilde{\mathbf{w}}_k)$
end
Assemble $\widehat{W}$ and invert scaling: $\widehat{W}_{i,:} = \widehat{W}_{i,:}/\beta_i$.
Clip: $\widehat{W} = \mathrm{clip}(\widehat{W}, -\tau, \tau)$.

Table 1: PTQ vs. QAT ($\Delta_0 = 1.5$ for all VQ settings; $\Delta_0 = 0.3$ for Babai PTQ). E and B denote Exact and Babai, respectively; M-Bits(s) represent MLP bits.

M-Bit(s)	QAT			PTQ				QAT			PTQ			
	Brevitas	VQ-E	VQ-B	Brevitas	VQ-E	VQ-B	QuIP	Brevitas	VQ-E	VQ-B	Brevitas	VQ-E	VQ-B	QuIP
	ReCon							IMDB						
1.00	Error	69.46	**87.92**	Error	8.27	12.91	57.03	Error	67.22	**80.60**	Error	31.35	41.94	79.11
1.58	-	**92.87**	89.55	-	47.20	10.19	-	-	**82.97**	81.84	-	78.91	55.36	-
2.00	92.19	**92.87**	92.30	32.08	46.39	16.21	87.91	84.75	84.05	82.28	63.06	83.72	63.46	**85.88**
2.32	-	**92.90**	91.39	-	66.45	14.71	-	-	83.19	81.93	-	78.67	68.81	-
2.58	-	**92.93**	91.81	-	71.39	19.78	-	-	82.59	81.28	-	83.59	77.78	-
2.81	-	**93.44**	91.53	-	63.12	40.60	-	-	82.79	82.88	-	81.15	77.60	-
3.00	**93.38**	93.23	93.13	48.42	66.12	45.31	92.24	85.49	83.12	82.73	82.19	83.03	80.16	**85.93**
3.17	-	93.24	91.72	-	65.02	44.28	-	-	84.43	82.55	-	84.38	82.32	-
4.00	92.95	**93.13**	92.22	90.90	65.25	56.50	92.89	85.14	84.03	82.03	84.89	82.75	84.81	**85.63**
	AntShield							SMS Spam						
1.00	Error	68.04	**87.91**	Error	0.97	29.68	71.74	Error	78.10	**90.55**	Error	0.00	13.61	84.62
1.58	-	92.09	91.23	-	33.14	23.21	-	-	**93.65**	95.24	-	86.55	44.14	-
2.00	91.67	**92.59**	91.31	17.45	52.18	27.97	83.28	**97.30**	94.95	95.95	0.00	74.31	75.95	96.60
2.32	-	**92.48**	90.55	-	61.14	34.98	-	-	96.00	94.52	-	76.41	83.85	-
2.58	-	**92.73**	92.06	-	57.06	39.35	-	-	95.36	95.33	-	65.21	85.29	-
2.81	-	**92.82**	91.46	-	59.10	33.25	-	-	97.28	95.02	-	65.49	77.04	-
3.00	**92.92**	92.33	92.11	72.79	56.73	24.40	71.75	96.30	**96.95**	95.62	96.00	68.98	85.30	95.68
3.17	-	92.25	90.66	-	29.86	15.13	-	-	96.64	92.51	-	66.67	76.36	-
4.00	**93.12**	92.83	91.90	77.88	52.68	54.24	92.34	96.32	94.59	95.21	**96.62**	62.08	67.42	96.00

4. Experiments and Results

We evaluate our quantization framework on MLP and pre-trained BERT, comparing SQ and VQ under both PTQ and QAT across multiple bit-widths. Experiments span four text-classification datasets: ReCon and AntShield (PII detection [2, 28–31]), IMDB sentiment, and SMS Spam. Only weights are quantized; activations remain FP32. We study two base scales, $\Delta_0 \in \{0.3, 1.5\}$, with $q \in [2, 8]$ and $M \in [1, 4]$. Additional results for higher bit budgets (> 4) are provided in the *Appendix*. Additional results, including (i) higher-bit settings (> 4 bits), (ii) another PTQ baseline (QuiP#) (iii) experiments on alternative lattice structure, (iv) multi-class classification, and (v) activation quantization using ACIQ, are reported in the *Appendix* to assess the generality of the proposed approach, with encouraging results and new insights. Note: M-Bit(s) refer to bit-width of MLP.

4.1. QAT vs. PTQ

Table 1 compares QAT and PTQ in the 1–4 bit regime, reporting the best score when multiple configurations match a given bit budget in case of VQ (eg, 2b has (2,2) and (4,1)). A key advantage of lattice-based VQ is its ability to operate at fractional bit-widths, yielding performance comparable to or better than integer-bit baselines. For example, at 1.58 bits, VQ-QAT achieves 92.87 on ReCon and 82.97 on IMDB, closely matching or exceeding the corresponding 2-bit performance, while no PTQ baseline achieves reliable accuracy at these bit-widths.

At 1 bit, PTQ methods largely fail across datasets, whereas VQ-QAT remains stable, achieving 87.92 on ReCon, 87.91 on AntShield, and 90.55 on SMS Spam. In the 2–4 bit range, QAT either closely matches or exceeds strong PTQ baselines in majority of cases. On IMDB, QuIP remains competitive, but QAT closely matches its performance (e.g., 85.49 vs. 85.93 at 3 bits).

Overall, except on IMDB where QuIP is strong, QAT in majority of cases matches or outperforms PTQ across datasets. Allowing fractional bit-widths enables VQ-QAT to achieve near–integer-bit performance at lower effective precision, making lattice-based QAT a more flexible and reliable choice in the low-bit regime.

4.2. Scalar vs. Vector Quantization

Table 2 compares SQ and lattice-based VQ across datasets. In the 1–1.58 bits regime, SQ consistently fails to train, whereas VQ remains stable and achieves strong performance across all datasets, establishing VQ as the only viable option under such constraints.

At 2 bits, overload plays a critical role. Configurations with q=2 suffer from very high overload ($\approx$ 60–80%) and reduced performance for VQ-Exact. In contrast, VQ configurations with reduced overload consistently achieve high F1 scores ($\approx$92–93 on ReCon and AntShield), indicating that

Table 2: Comparison of SQ and VQ with Δ_0=1.5. Here, OL represents overload. Here, M-Bits(s) represent MLP bits.

M-Bit(s)	SQ	VQ-Exact		VQ-Babai		SQ	VQ-Exact		VQ-Babai	
		F1	OL%	F1	OL%		F1	OL%	F1	OL%
	ReCon (FP-93.93)					**IMDB** (FP-85.59)				
1.00b(2,1)	Error	69.46	0.02	87.92	0	Error	67.22	0.00	80.60	0
1.58b(3,1)		**92.87**	2.25	89.55	0		**82.97**	2.62	81.84	0
2.00b(2,2)	90.87	78.79	60.81	92.30	0	83.23	83.03	61.42	82.28	0
2.00b(4,1)		92.87	0.02	92.30	0		**84.05**	0.02	82.28	0
2.32b(5,1)		92.90	0.00	91.39	0		83.19	0.01	81.93	0
2.58b(6,1)		92.93	0.01	91.81	6.98		82.59	0.01	81.28	6.60
2.81b(7,1)		**93.44**	0.00	91.53	0		82.79	0.01	82.88	0
3.00b(2,3)	**93.49**	79.56	79.14	93.13	0	83.98	83.12	79.88	82.73	0
3.00b(8,1)		93.23	0.00	91.25	0		83.12	0.00	81.92	0
3.17b(3,2)		93.24	1.00	91.72	0		**84.43**	1.28	82.55	0
4.00b(2,4)	92.84	80.82	83.71	92.22	0	83.86	84.03	84.97	82.03	0
4.00b(4,2)		**93.13**	0.40	92.22	0		83.42	0.39	82.03	0
	AntShield (FP: 93.22)					**SMS Spam** (FP: 95.65)				
1.00b(2,1)	Error	68.04	0.04	87.91	0	Error	78.10	0.01	90.55	0
1.58b(3,1)		**92.09**	2.38	91.23	0		93.65	2.25	**95.24**	0
2.00b(2,2)	90.83	78.25	60.68	91.31	0	95.59	90.41	59.01	95.95	0
2.00b(4,1)		92.59	0.01	91.31	0		94.95	0.01	95.95	0
2.32b(5,1)		92.48	0.00	90.55	0		96.00	0.01	94.52	0
2.58b(6,1)		92.73	0.01	92.06	6.716		95.36	0.02	95.33	6.333
2.81b(7,1)		**92.82**	0.00	91.46	0		**97.28**	0.01	95.02	0
3.00b(2,3)	91.93	77.21	78.64	92.11	0	94.98	90.10	76.70	94.31	0
3.00b(8,1)		**92.33**	0.01	91.77	0		**96.95**	0.01	95.62	0
3.17b(3,2)		92.25	1.14	90.66	0		96.64	1.42	92.51	0
4.00b(2,4)	92.51	79.12	84.03	91.90	0	95.62	92.36	81.96	95.21	0
4.00b(4,2)		**92.83**	0.35	91.90	0		94.59	0.30	95.21	0

accuracy improves as overload is controlled. In this regime, VQ variants match or exceed SQ in most cases. At 3 bits, SQ becomes competitive. Beyond 3 bits, performance differences narrow further, but VQ continues to exhibit more consistent behavior across configurations due to superior overload control. Overall, lattice-based VQ under QAT outperforms SQ in the majority of settings, particularly below 4 bits, highlighting that effective overload control—especially avoiding q=2—is crucial for robust low-bit quantization.

Role of (q, M) in VQ: We observe trends consistent with the geometric behavior described in [4]: for a fixed bit budget, configurations with larger q and smaller M are more stable and accurate. For example, at 4 bits, $(2, 4)$ attains only 80.82 F1, whereas $(4, 2)$ reaches 93.13 in VQ Exact (ReCon). When M is large, and q is small, residual corrections accumulate error rather than reducing it, degrading performance. From an implementation standpoint, $(4, 2)$ is also attractive as it fits into a compact LUT, unlike alternatives such as $(16, 1)$, which trade higher memory for minimal compute. At higher bit-widths, very large q offers diminishing returns as performance saturates. Overall, accuracy favors larger q and smaller M, while the final choice reflects a trade-off between memory and compute.

4.3. BERT quantization

We extend our study to linear layers of BERT (in our case 73 layers in bert-base-uncased). In all experiments, we start from the standard pretrained checkpoint and perform end-to-end QAT, rather than training BERT from scratch. This approach is computationally feasible while still allowing the quantizer and model to co-adapt during fine-tuning. We adopt mixed-precision training, meaning attention softmax, layer norms, and non-linearities remain in FP16/FP32, while the linear projections are quantized using VQ.

Table 3 reports results for four settings: (i) end-to-end QAT of both BERT and MLP ($\mathbf{B^{QE/QB}M^{QE/QB}}$), (ii) FP32 BERT with PTQ ($\mathbf{B^{F}M^{PE/PB}}$), and (iii) FP32 BERT with QAT MLP ($\mathbf{B^{F}M^{QE/QB}}$). FP32 BERT with PTQ MLP suffers severe degradation, collapsing to near-random performance on ReCon and AntShield, but catches up at 3 bits for IMDB and SMS Spam datasets. End-to-end QAT significantly mitigates this degradation, and performance recovers substantially across all bit-widths. At 2–3 bits, QAT achieves F1 scores close to FP32 in the majority of datasets, demonstrating that the lattice-based quantizer can be stably integrated into BERT when co-adapted during training.

Table 3: Comparison of BERT FP32 and quantization performance. Here, ^{PE}:PTQ with Exact, ^{PB}:PTQ with Babai, ^{QE}:QAT with Exact, ^{QB}:QAT with Babai, ^{FP}: FP32. For Exact Δ_0=1.5 and for babai Δ_0=0.3. Here, *Config* represent Bert and MLP bits.

Config	FP32	$B^{QE}M^{QE}$	$B^{QB}M^{QB}$	$B^{F}M^{PE}$	$B^{F}M^{PB}$	$B^{F}M^{QE}$	$B^{F}M^{QB}$	FP32	$B^{QE}M^{QE}$	$B^{QB}M^{QB}$	$B^{F}M^{PE}$	$B^{F}M^{PB}$	$B^{F}M^{QE}$	$B^{F}M^{QB}$
	ReCon							**IMDB**						
1b(2,1)		68.25	38.88	8.27	12.91	69.46	70.22		82.37	82.35	31.35	41.94	67.22	72.39
2b(4,1)	93.93	85.29	67.32	46.39	16.21	92.87	81.70	85.59	78.54	66.67	83.72	62.32	84.05	83.32
3b(8,1)		87.65	68.02	66.12	45.31	93.23	91.43		87.12	78.33	83.03	80.16	83.12	81.37
	AntShield							**SMS Spam**						
1b(2,1)		71.33	63.51	0.97	29.68	68.04	72.72		0	23.05	0.00	13.61	78.10	73.08
2b(4,1)	93.22	89.17	57.06	52.18	21.72	92.59	84.86	95.65	96.05	7.55	74.31	75.95	94.95	93.65
3b(8,1)		90.95	27.85	56.73	24.40	92.33	91.90		98.65	26.57	68.98	83.38	96.95	94.20

If we quantize the entire model end-to-end, it results in a substantial reduction in memory footprint. This trade-off is particularly attractive in practice, as the accuracy drop remains modest relative to the significant compression gains achieved. Since quantization is introduced only during fine-tuning, the approach remains compatible with standard training pipelines and avoids the prohibitive cost of quantization-aware pretraining. Methods such as QLoRA-style quantized pretraining may further improve alignment between the model and the quantizer; is part of future work.

5. Conclusion

This work studied lattice-based VQ as a mechanism for enabling stable QAT at extremely low bit-widths. Across both PTQ and QAT settings, our results demonstrate that structured lattice projections offer a decisive advantage over SQ when operating under aggressive compression. In particular, our experiments show that <2 bits, lattice-based VQ is the only approach that consistently trains successfully and preserves meaningful task performance. In this regime, SQ and existing PTQ methods frequently collapse or exhibit severe instability, whereas lattice-based QAT remains robust. This advantage arises from exploiting geometric structure across weight blocks, which enables more efficient use of the limited bit budget and substantially reduces overload. We further demonstrate that quantization behavior is governed not only by nominal bit-width but by lattice parameters, including the choice of (q, M), base scale Δ_0, and decoding strategy. These parameters directly control overload frequency and effective dynamic range, providing practitioners with explicit levers to balance accuracy, compression, and compute. When overload is kept negligible, the fused lattice projection effectively collapses to a single nearest-lattice operation, and hierarchical refinement can be omitted without loss of fidelity. We also explored several extensions to assess the generality of the proposed framework, including exploring D_4 lattice in addition to E_8, preliminary studies on activation quantization, and evaluations on a multi-class classification dataset. While these results are encouraging, they are not exhaustive and are therefore reported in the *Appendix*.

Finally, we show that lattice-based QAT extends beyond small models and can be applied to transformers such as BERT, where direct PTQ at very low precision is highly unstable. End-to-end QAT enables substantial recovery of accuracy with controlled degradation, making lattice-based VQ a practical and scalable tool for deploying deep learning models at extremely low precision.

Limitations and Future Scope. Despite encouraging results, several limitations remain. The proposed lattice-based VQ framework is not well suited to PTQ-only scenarios, where methods such as QUIP and QUIP# are more appropriate. Broader validation across larger benchmarks (e.g., MMLU), regression tasks, and additional modalities is required to more comprehensively assess generality. Moreover, aggressive joint quantization, particularly combinations of low weight and low activation precision, exhibits instability and warrants deeper analysis. Several BERT components, including attention softmax, LayerNorm, and embedding layers, are currently kept in full precision, leaving further opportunities for end-to-end compression. In addition, the current implementation focused on correctness over efficiency, and the investigation prioritized scientific inquiry over performance engineering; key components such as lattice operations and LUT-based support remain unaddressed as a result.

References

[1] Van Minh Nguyen, Cristian Ocampo-Blandon, Aymen Askri, Louis Leconte, and Ba-Hien Tran. Bold: Boolean logic deep learning. *Advances in Neural Information Processing Systems*, 37:61912–61962, 2024.

[2] Rishika Kohli, Shaifu Gupta, Manoj Singh Gaur, and Soma S Dhavala. Learning to detect PII: Tabular vs. Document classification models for network traffic analysis. *Journal of Information Security and Applications*, 94:104196, 2025.

[3] Huabin Diao, Gongyan Li, Shaoyun Xu, Chao Kong, and Wei Wang. Attention round for post-training quantization. *Neurocomputing*, 565:127012, 2024.

[4] Iris Kaplan and Or Ordentlich. High-rate nested-lattice quantized matrix multiplication with small lookup tables, 2025. URL `https://arxiv.org/abs/2505.13164`.

[5] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. Bitnet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*, 2023.

[6] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients, 2018. URL `https://arxiv.org/abs/1606.06160`.

[7] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[8] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[9] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[10] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[11] Qing Jin, Linjie Yang, and Zhenyu Liao. Towards efficient training for neural network quantization. *arXiv preprint arXiv:1912.10207*, 2019.

[12] Giuseppe Franco, Alessandro Pappalardo, and Nicholas J Fraser. Xilinx/brevitas, 2025. URL `https://doi.org/10.5281/zenodo.3333552`.

[13] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in neural information processing systems*, 32, 2019.

[14] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018. IEEE, 2019.

[15] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1325–1334, 2019.

[16] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International conference on machine learning*, pages 7197–7206. PMLR, 2020.

[17] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021.

[18] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36:4396–4429, 2023.

[19] Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*, 2024.

[20] Rajarshi Saha, Naomi Sagan, Varun Srivastava, Andrea Goldsmith, and Mert Pilanci. Compressing large language models using low rank and low precision decomposition. *Advances in Neural Information Processing Systems*, 37:88981–89018, 2024.

[21] Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

[22] Ram Zamir. *Lattice coding for signals and networks: A structured coding approach to quantization, modulation, and multiuser information theory*. Cambridge University Press, 2014.

[23] John Conway and Neil Sloane. Fast quantizing and decoding and algorithms for lattice quantizers and codes. *IEEE Transactions on Information Theory*, 28(2):227–232, 2003.

[24] John Horton Conway and Neil James Alexander Sloane. *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media, 2013.

[25] Debargha Mukherjee and Sanjit K Mitra. Successive refinement lattice vector quantization. *IEEE Transactions on Image Processing*, 11(12):1337–1348, 2002.

[26] Guillaume Fuchs. Embedded voronoi codes for successive refinement lattice vector quantization. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5805–5809. IEEE, 2013.

[27] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. ACIQ: Analytical Clipping for Integer Quantization of neural networks. 2018.

[28] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Augustin Chaintreau Legout, and David Choffnes. ReCon: Revealing and controlling PII leaks in mobile network traffic. In *Proc. 14th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, pages 361–374, 2016.

[29] Anastasia Shuba, Evita Bakopoulou, Milad Asgari Mehrabadi, Hieu Le, David R. Choffnes, and Athina Markopoulou. Antshield: On-Device detection of personal information exposure. *CoRR*, abs/1803.01261, 2018.

[30] Rishika Kohli, Shreyas Chatterjee, Shaifu Gupta, and Manoj Singh Gaur. Tracking PII exfiltration: Exploring decision tree and neural network with explainable AI. In *2023 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 183–188. IEEE, 2023.

[31] Rishika Kohli, Shaifu Gupta, and Manoj Singh Gaur. A deep dive into relevant feature identification for unveiling PII leakage in smartphones. In *2024 International Conference on Signal Processing and Communications (SPCOM)*, pages 1–5. IEEE, 2024.

[32] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Optq: Accurate quan- tization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023.

6. LLM usage

LLMs were used as co-pilots during various stages of this work, including code generation, improving the clarity of the write-up, and assisting in the development and refinement of theoretical ideas. These tools supported the research process but did not replace the authors' own technical contributions or judgment. The authors take full responsibility for the accuracy, correctness, and originality of all content in this paper.

A. Appendix

Algorithm 3 Nearest-neighbor projection $Q_{\mathrm{E8}}(\cdot)$

Input: $x \in \mathbb{R}^8$ (or batch $x \in \mathbb{R}^{B\times 8}$)
Output: $y \in E_8$

Let $\mathbf{h} = (\frac{1}{2}, \ldots, \frac{1}{2})$.

$f \leftarrow \mathrm{custom\_round}(x)$; $y_0 \leftarrow \begin{cases} f, & \text{if } (\sum_i f_i) \bmod 2 = 0 \\ g(x), & \text{otherwise} \end{cases}$

$f' \leftarrow \mathrm{custom\_round}(x - \mathbf{h})$; $y_1 \leftarrow \begin{cases} f' + \mathbf{h}, & \text{if } (\sum_i f'_i) \bmod 2 = 0 \\ g(x-\mathbf{h}) + \mathbf{h}, & \text{otherwise} \end{cases}$

return $y = \arg\min\{\|x - y_0\|_2,\ \|x - y_1\|_2\}$.

Helper-1: $\mathrm{custom\_round}(\cdot)$ defined as $\mathrm{custom\_round}(v) = \lfloor v - \mathrm{sign}(v)\varepsilon + \frac{1}{2} \rfloor$, where $\varepsilon > 0$ is a tiny constant.

Helper-2: Let $f(x) = \mathrm{custom\_round}(x)$ and $k = \arg\max_j |x_j - f(x)_j|$. Then $g(x)$ equals $f(x)$ except at coordinate k, which is changed by ± 1, so that the parity of $\sum_i f(x)_i$ flips and ties are resolved consistently.

Algorithm 4 Babai projection onto E_8

Input: generator matrix $G \in \mathbb{R}^{8\times 8}$, inverse G^{-1}; input $x \in \mathbb{R}^8$ (or batch $x \in \mathbb{R}^{B\times 8}$).
Output: $\tilde{y} \approx \Pi_{E_8}(x)$.

Lattice coordinates $z \leftarrow x(G^{-1})^\top$.

Integer rounding $\hat{z} \leftarrow \mathrm{round}(z)$. `// standard rounding`

Map back to $\mathbb{R}^8$ $\tilde{y} \leftarrow \hat{z}G^\top$.

Algorithm 5 Scalar quantization $Q_{\mathrm{SQ}}(\cdot)$

Input: pre-scaled tensor x; radix q; levels M.
Output: x_{q} (quantized tensor).

Effective bit-width $b = \lfloor M \log_2(q) \rfloor$.

Integer limits $q_{\max} = 2^{b-1} - 1, \qquad q_{\min} = -q_{\max}$.

Round and clip. $x_{\mathrm{q}} = \mathrm{clip}\big(\mathrm{round}(x),\ q_{\min}, q_{\max}\big)$.

A.1. Nearest-neighbor projection and scalar quantization

The exact nearest-neighbor projection onto the E_8 lattice (Algorithm 3) is computed by explicitly evaluating candidates from its two structured cosets, D_8 and $D_8 + (0.5)^8$. For each coset, a rounded candidate is constructed and corrected to satisfy the even-parity constraint via a minimal coordinate adjustment, and the candidate with the smallest Euclidean distance to the input is selected, guaranteeing the true nearest lattice point. As a lower-complexity alternative, we also consider the Babai projection (Algorithm 4), which operates in the coordinate system induced by a non-orthogonal generator matrix, applies integer rounding in that basis, and maps back to $\mathbb{R}^8$.

Note: Exact and Babai quantization techniques rely on two fundamentally different mechanisms. Exact searches over the two structured cosets of E_8 (D_8 and $D_8 + (0.5)^8$) guarantee the true nearest lattice point for E_8. Babai instead works in the coordinate system of a non-orthogonal generator matrix G and applies the nearest-plane rule. Because G is not orthogonal, Babai's decision regions do not match the true Voronoi regions of E_8, so it provides a faster but approximate alternative. It can be considered a different quantizer altogether. When G is orthogonal, and no overloading occurs, both Exact and Babai are equivalent.

Algorithm 6 Overload detection via hierarchical E_8 encoding

Input: quantized blocks from vector quantization $\widehat{\mathbf{W}}^{\text{blocks}} \in \mathbb{R}^{N\times 8}$ (Step 5 of Alg. 1);
radix $q \in \mathbb{N}$, levels $M \in \mathbb{N}$;
(optional) scale $\beta > 0$ used in the encoder; overload tolerance $\varepsilon > 0$.
Output: number of overloaded blocks N_{over} and overload flags $o_k \in \{0,1\}$ for $k = 1, \dots, N$.

Block indexing $\widehat{\mathbf{W}}^{\text{blocks}} = \begin{bmatrix} (\widehat{\mathbf{w}}^{(1)})^\top \\ \vdots \\ (\widehat{\mathbf{w}}^{(N)})^\top \end{bmatrix}, \qquad \widehat{\mathbf{w}}^{(k)} \in \mathbb{R}^8.$

for $k = 1, \dots, N$ **do**
- $\mathbf{x}_0^{(k)} \leftarrow \widehat{\mathbf{w}}^{(k)}/\beta$ — `// Match encoder input scaling`
- **for** $m = 1$ **to** M **do**
  - $\mathbf{u}_m^{(k)} \leftarrow Q_{\text{E8}}(\mathbf{x}_{m-1}^{(k)})$ — `// Nearest` E_8 `lattice point`
  - $\mathbf{z}_m^{(k)} \leftarrow \text{encode\_coords}(\mathbf{u}_m^{(k)}, q)$ — `// Lattice coords (base-`q` digits)`
  - $\mathbf{b}_m^{(k)} \leftarrow \mathbf{z}_m^{(k)} \bmod q$ — `// Stored base-`q` digits in` $[0, q-1]^8$
  - $\mathbf{x}_m^{(k)} \leftarrow \mathbf{u}_m^{(k)}/q$ — `// Hierarchical state for next level`
- **end**
- $\mathbf{r}^{(k)} \leftarrow Q_{\text{E8}}(\mathbf{x}_M^{(k)})$ — `// Residual after` M `levels`
- $o_k \leftarrow \mathbf{1}\left[\|\mathbf{r}^{(k)}\|_2 > \varepsilon\right]$ — `// Overload if final residual is non-zero`

end

$N_{\text{over}} \leftarrow \sum_{k=1}^{N} o_k$ — `// Aggregate overload statistics`

Algorithm 5 describes SQ, which can be viewed as a special case of VQ operating independently on each coordinate. Given a radix q and M hierarchical levels, the effective bit-width is $b = \lfloor M \log_2(q) \rfloor$, and quantization is performed by element-wise rounding followed by clipping to the representable integer range. Unlike lattice-based VQ, SQ does not exploit inter-dimensional structure, but it provides a useful baseline for isolating the gains from structured VQ.

A.2. Overload detection

To detect overload, we apply the hierarchical E_8 encoder to the quantized blocks $\widehat{\mathbf{W}}^{\text{blocks}}$. Each block is recursively projected and rescaled across M levels, producing a final residual vector. If this residual is non-zero (or exceeds a small tolerance), the block is marked as overloaded, indicating that its magnitude exceeds the dynamic range supported by (q, M). Algorithm 6 summarizes this process and returns both per-block overload flags and aggregate overload statistics. This post-hoc overload analysis is used only for evaluation and characterization; it does not affect forward or backward passes during training.

A.3. Heuristics for dynamic range of quantizer:

Let $L \subset \mathbb{R}^d$ be a full-rank lattice and consider a hierarchical nested lattice quantizer (HNLQ) with radix $q \geq 2$, M digit levels, and finest step size $\Delta_0 > 0$. Each scalar codeword generated by the HNLQ can be written as $y_{\text{code}} = \Delta_0 \sum_{m=0}^{M-1} d_m q^m, \qquad d_m \in \mathcal{D}$, where $\mathcal{D}$ is a finite digit set. If the digits are chosen so that the scalar hierarchy covers a symmetric interval around zero, then the effective representable range can be approximated as $y_{\text{code}} \in [-Y_{\max}, Y_{\max}], \qquad Y_{\max} \approx \frac{\Delta_0}{2}\big(q^M - 1\big)$.

A.4. Experimental Details

For all the results, we use four datasets: i) ReCon [28], ii) AntShield [29], iii) IMDB, and iv) SMS Spam. We construct 80/10/10 train/validation/test splits with stratification over the binary label.

A.4.1. Experimental Setup

We evaluate our unified quantization framework on two architectures: MLP and BERT.

MLP pipeline: We first extract fixed full-precision (FP32) sentence embeddings using the pre-trained `bert-base-uncased` encoder and then train a shallow quantized MLP on top of these 768-dimensional embeddings. The MLP consists of a mean-centering layer, a fully connected layer with

512 hidden units followed by a ReLU activation, and a final linear layer that outputs a single logit for binary classification. Only the MLP weights are quantized by replacing the first fully connected linear layer with a quantized version using the fused setup explained in Section 3.1.3; the BERT encoder remains frozen in FP32 throughout.

We train the MLP with the Adam optimizer (learning rate 10^{-3}), batch size 64, and binary cross-entropy with logits loss. The model is trained for up to 200 epochs. This simple architecture isolates the effect of lattice-based quantization on the classifier while keeping the upstream BERT representations fixed.

BERT end-to-end pipeline. We start from the pretrained `bert-base-uncased` encoder and attach the same shallow MLP head used in above pipeline. All `nn.Linear` layers inside BERT and in the MLP head are replaced with our fused setup explained in Section 3.1.3.

Input texts are tokenized with a maximum sequence length of 128. The entire model (quantized BERT encoder plus MLP head) is trained end-to-end using AdamW with a learning rate of 2×10^{-5}, a batch size of 32, and a binary cross-entropy with logits loss. We train for up to 25 epochs, selecting the best checkpoint according to validation F1. This setting directly tests whether lattice-based VQ can be pushed into all linear layers of a transformer while preserving downstream binary classification performance.

A.4.2. Integration into the Quantization Pipeline

The proposed scalar and vector quantizers are integrated into the MLP classifier and BERT through a fused quantized linear operator. Each fully connected layer $\mathrm{FC}(x) = Wx + b$ is replaced by a quantized operator $\mathrm{FC_Q}(x) = \widehat{W}\,x + b$ where $\widehat{W}$ is obtained using either fused E_8 VQ (Algorithm 1) or SQ (Algorithm 2). In both PTQ and QAT settings, only the weights are quantized.

In **PTQ setting**, $\widehat{W}$ is computed once using the learned full-precision weights, and the network is evaluated without any further training. Dynamic ranges are estimated from the trained model, and no gradient updates are involved. Whereas in **QAT setting**, quantization is applied on every forward pass. The network uses $\widehat{W}$ for inference, but gradients are applied to the underlying full-precision parameters W through STE.

In VQ, the fused implementation ensures that scaling, block partitioning, projection, reconstruction, and clipping occur within a single differentiable operator, improving numerical stability and reducing memory transfers during training.

A.4.3. Hyperparameter Choice

Lattice quantizer is controlled by the hierarchical parameters (q, M), the base radius Δ_0, and the row-safety constant C_b. As discussed in Section 3.1.3, (q, M) determine the effective representable radius $Y_{\max} = \Delta_0(q^M - 1)/2$ and hence the nominal bit budget. In practice, Δ_0 and C_b act as coarse and fine controls on how aggressively each row is scaled before E_8 projection. Row-safety constant is fixed to $C_b = 5$, which roughly corresponds to mapping weights into a 5σ range per row. This choice limits overload events while avoiding overly conservative scaling. The base radius Δ_0 is set to 1.5 and 0.3.

A.4.4. Additional Experimental Results

a) PTQ vs. QAT. Table 4 presents a comparison across 1–12 bits, while Table 5 reports fractional bit-width results for VQ variants. At $\geq$4 bits, the performance gap between QAT and PTQ narrows. QuIP is strongest in most configurations on IMDB and ReCon, while Brevitas remains competitive on ReCon, AntShield, and SMS Spam. In contrast, at ultra-low precision (1 and 1.5 bits), PTQ methods either fail or degrade significantly across datasets, whereas our lattice-based VQ-QAT variants consistently achieve strong and stable performance, establishing them as the most effective approach in this regime.

Table 4: PTQ vs. QAT ($\Delta_0 = 1.5$ for all settings and 0.3 for Babai PTQ). For each bit-width, we use the nearest (q, M) setting and report the best score when multiple configurations match the bit budget.

M-Bit(s)	QAT			PTQ				QAT			PTQ			
	Brevitas	**VQ-E**	**VQ-B**	**Brevitas**	**VQ-E**	**VQ-B**	**QuIP**	**Brevitas**	**VQ-E**	**VQ-B**	**Brevitas**	**VQ-E**	**VQ-B**	**QuIP**
	ReCon							**IMDB**						
1	Error	69.46	**87.92**	Error	8.27	12.91	57.03	Error	67.22	**80.60**	Error	31.35	41.94	79.11
2	92.19	**92.87**	92.30	32.08	46.39	16.21	87.91	84.75	84.05	82.28	63.06	83.72	63.46	**85.88**
3	**93.38**	93.23	93.13	48.42	66.12	45.31	92.24	85.49	83.12	82.73	82.19	83.03	80.16	**85.93**
4	92.95	**93.13**	92.22	90.90	65.25	56.50	92.89	85.14	84.03	82.03	84.89	82.75	84.81	**85.63**
5	93.32	93.54	90.47	92.58	59.40	52.09	**93.67**	85.10	82.83	81.27	85.61	83.84	83.43	**85.53**
6	**94.32**	93.46	92.65	93.90	69.42	71.01	94.16	85.47	83.06	83.08	85.76	82.76	83.47	**85.59**
7	93.20	92.71	91.11	93.56	53.06	59.16	**93.95**	85.40	83.12	80.64	85.38	83.40	83.91	**85.57**
8	92.37	93.23	92.83	**93.99**	64.27	64.85	93.93	84.77	83.82	83.32	85.54	78.32	82.76	**85.64**
9	**93.96**	93.23	91.92	93.84	60.45	52.65	**93.96**	85.53	82.14	81.22	**85.57**	84.00	81.15	85.54
10	**93.91**	93.22	91.65	93.86	74.80	41.45	**93.91**	84.94	83.16	81.49	85.54	68.67	85.29	**85.59**
11	93.56	92.99	91.10	**93.93**	43.38	45.97	93.91	85.34	82.52	80.84	**85.59**	84.62	80.11	**85.59**
12	93.82	93.06	92.12	93.94	41.65	58.59	**93.96**	85.34	81.67	82.87	**85.59**	80.58	82.97	**85.59**
	AntShield							**SMS Spam**						
1	Error	68.04	**87.91**	Error	0.97	29.68	71.74	Error	78.10	**90.55**	Error	0.00	13.61	84.62
2	91.67	**92.59**	91.31	17.45	52.18	27.97	83.28	**97.30**	94.95	95.95	0.00	74.31	75.95	96.60
3	**92.92**	92.33	92.11	72.79	56.73	24.40	71.75	96.30	**96.95**	95.62	96.00	68.98	85.30	95.68
4	**93.12**	92.83	91.90	77.88	52.68	54.24	92.34	96.32	94.59	95.21	**96.62**	62.08	67.42	96.00
5	**93.60**	92.62	91.38	90.90	61.70	55.87	92.80	**96.64**	96.32	96.27	96.00	70.62	81.99	96.00
6	93.38	92.22	92.39	92.95	53.27	53.98	**93.42**	96.64	**96.97**	94.88	95.97	78.01	79.89	95.97
7	**93.38**	92.73	90.77	93.10	49.13	44.15	93.30	**96.64**	96.30	94.67	95.65	69.46	66.07	96.00
8	**93.51**	91.96	91.96	93.20	47.22	59.10	93.14	96.64	**97.63**	94.88	95.65	73.40	68.04	95.65
9	93.22	91.78	91.77	93.22	52.35	50.97	**93.25**	**96.64**	96.60	94.00	95.65	78.84	77.49	95.65
10	**93.33**	92.05	91.02	93.22	57.52	54.24	93.25	**96.64**	95.65	93.65	95.65	70.28	75.90	95.65
11	**93.61**	91.97	91.11	93.20	56.59	51.27	93.15	**96.32**	95.68	93.33	95.65	76.68	82.86	95.65
12	93.02	92.03	91.78	93.20	56.45	51.01	**93.18**	**96.64**	94.70	93.24	95.65	80.00	66.82	95.65

b) VQ vs. SQ. Table 5 reports a detailed comparison of SQ and lattice-based VQ across integer and fractional bit-widths. At 1-1.5 bits precision, SQ fails to train on both datasets, whereas VQ remains stable. As bit-width increases beyond 3 bits, the performance gap between SQ and VQ narrows. In Table 5, results are summarized by bit-ranges (e.g., 1–1.58,b, 2.x,b, 3.x,b), and for each range we report the best-performing configuration across both SQ and VQ, rather than restricting comparisons to exact integer bit-widths. On ReCon, VQ-Exact achieves the best performance in most configurations, with SQ surpassing VQ only in a few isolated cases. On IMDB, VQ-Exact remains superior across nearly all bit-widths, except for a single configuration around 7.75 bits. In AntShield, SQ begins to outperform VQ at higher precisions (> 8, bits), while on SMS Spam, this trend appears only beyond 10, bits. Overall, VQ—particularly Exact projection—remains the dominant approach across the majority of datasets and bit-widths.

c) Computational overhead of QAT: To quantify the training-time cost introduced by QAT, we measure the per-epoch wall-clock overhead of our fused QAT relative to a full-precision (FP32) baseline. Figures 1 show results for all four datasets. Across all experiments, VQ incurs additional computation during the forward pass due to block partitioning, scaling, and lattice projection. However, the overhead remains modest at roughly 1.31–1.50× the speed of FP32 for VQ Babai setting but is 4-5× for VQ Exact.

The trend is similar across all datasets, demonstrating that lattice projection dominates the additional cost, while dataset size and model depth have minimal impact. These results show that although QAT with structured VQ introduces measurable overhead, the cost remains practically manageable, especially considering the accuracy gains achieved at low precision.

d) Effect of warm-up

Since lattice-based PTQ exhibits poor performance at low bit-widths due to the absence of gradient adaptation, we examine whether an initial FP32 warm-up can mitigate this instability. Specifically, the model is first trained in full precision for 5 epochs and then switched to quantized training. As shown in Figure 2, the transition from FP32 to quantized weights causes a sharp loss spike, as this switch is effectively equivalent to applying PTQ at the transition point—a sudden projection of weights onto the quantization lattice. This PTQ-like shock explains why VQ-PTQ without fine-tuning performs poorly and why QAT is required for recovery.

Table 5: Comparison of SQ and VQ with Δ_0=1.5.

M-Bit(s)	SQ	VQ-Exact		VQ-Babai		SQ	VQ-Exact		VQ-Babai	
		F1	OL%	F1	OL%		F1	OL%	F1	OL%
	ReCon (FP-93.93)					**IMDB** (FP-85.59)				
1.00b(2,1)	Error	69.46	0.02	87.92	0	Error	67.22	0.00	80.60	0
1.58b(3,1)		**92.87**	2.25	89.55	0		**82.97**	2.62	81.84	0
2.00b(2,2)	90.87	78.79	60.81	92.30	0	83.23	83.03	61.42	82.28	0
2.00b(4,1)		92.87	0.02	92.30	0		**84.05**	0.02	82.28	0
2.32b(5,1)		92.90	0.00	91.39	0		83.19	0.01	81.93	0
2.58b(6,1)		92.93	0.01	91.81	6.98		82.59	0.01	81.28	6.60
2.81b(7,1)		**93.44**	0.00	91.53	0		82.79	0.01	82.88	0
3.00b(2,3)	**93.49**	79.56	79.14	93.13	0	83.98	83.12	79.88	82.73	0
3.00b(8,1)		93.23	0.00	91.25	0		83.12	0.00	81.92	0
3.17b(3,2)		93.24	1.00	91.72	0		**84.43**	1.28	82.55	0
4.00b(2,4)	92.84	80.82	83.71	92.22	0	83.86	84.03	**84.97**	82.03	0
4.00b(4,2)		**93.13**	0.40	92.22	0		83.42	0.39	82.03	0
4.64b(5,2)		92.94	0.20	91.50	0		82.14	0.24	81.36	0
4.75b(3,3)		92.54	3.52	90.80	0		82.92	4.46	81.55	0
5.17b(6,2)	93.05	**93.54**	0.19	90.47	7.62	82.79	**82.83**	0.22	81.27	7.58
5.61b(7,2)		92.88	0.17	90.10	0		82.49	0.20	82.04	0
6.00b(4,3)	92.91	92.84	0.78	92.65	0	82.38	83.06	0.78	83.08	0
6.00b(8,2)		**93.46**	0.08	92.48	0		82.28	0.10	82.13	0
6.34b(3,4)		93.24	5.04	90.44	0		**83.81**	6.14	81.80	0
6.97b(5,3)		92.71	0.36	91.11	0		83.12	0.37	80.64	0
7.75b(6,3)	**93.69**	93.29	0.36	91.93	8.08	**82.70**	82.68	0.34	82.21	8.06
8.00b(4,4)	93.12	**93.23**	0.76	92.83	0	83.37	**83.82**	0.80	83.32	0
8.42b(7,3)		92.67	0.26	91.19	0		81.34	0.26	81.79	0
9.00b(8,3)	93.17	93.23	0.14	91.92	0	82.85	82.14	0.13	81.22	0
9.29b(5,4)		**93.63**	0.42	90.27	0		**83.60**	0.44	81.36	0
10.34b(6,4)	93.11	**93.22**	0.36	91.65	8.10	81.32	**83.16**	0.39	81.49	8.18
11.23b(7,4)	92.90	**92.99**	0.27	91.10	0	82.25	**82.52**	0.28	80.84	0
12.00b(8,4)	**93.16**	93.06	0.13	92.12	0	82.46	81.67	0.13	**82.87**	0
	AntShield					**SMS Spam**				
1.00b(2,1)	Error	68.04	0.04	87.91	0	Error	78.10	0.01	90.55	0
1.58b(3,1)		**92.09**	2.38	91.23	0		93.65	2.25	**95.24**	0
2.00b(2,2)	90.83	78.25	60.68	91.31	0	95.59	90.41	59.01	95.95	0
2.00b(4,1)		92.59	0.01	91.31	0		94.95	0.01	95.95	0
2.32b(5,1)		92.48	0.00	90.55	0		96.00	0.01	94.52	0
2.58b(6,1)		92.73	0.01	92.06	6.716		95.36	0.02	95.33	6.333
2.81b(7,1)		**92.82**	0.00	91.46	0		**97.28**	0.01	95.02	0
3.00b(2,3)	91.93	77.21	78.64	92.11	0	94.98	90.10	76.70	94.31	0
3.00b(8,1)		**92.33**	0.01	91.77	0		**96.95**	0.01	95.62	0
3.17b(3,2)		92.25	1.14	90.66	0		96.64	1.42	92.51	0
4.00b(2,4)	92.51	79.12	84.03	91.90	0	95.62	92.36	81.96	95.21	0
4.00b(4,2)		**92.83**	0.35	91.90	0		94.59	0.30	95.21	0
4.64b(5,2)		92.69	0.15	91.40	0		**96.64**	0.25	94.98	0
4.75b(3,3)		92.52	3.42	90.89	0		96.62	4.49	94.00	0
5.17b(6,2)	**92.62**	**92.62**	0.19	91.38	7.741	94.39	**96.32**	0.23	96.27	7.467
5.61b(7,2)		92.12	0.16	90.88	0		94.43	0.18	93.02	0
6.00b(4,3)	92.62	92.22	0.67	92.39	0	96	96.30	0.70	92.31	0
6.00b(8,2)		92.10	0.08	91.36	0		**96.97**	0.10	94.88	0
6.34b(3,4)		92.29	4.71	90.33	0		96.93	6.32	93.11	0
6.97b(5,3)		**92.73**	0.42	90.77	0		96.30	0.46	94.67	0
7.75b(6,3)	92.27	**92.45**	0.35	91.24	8.134	95.33	**95.97**	0.39	94.31	7.642
8.00b(4,4)	**92.54**	91.96	0.78	91.96	0	95.30	**97.63**	0.81	94.88	0
8.42b(7,3)		92.51	0.24	90.89	0		95.97	0.25	93.07	0
9.00b(8,3)	**91.98**	91.78	0.13	91.77	0	95.33	**96.60**	0.12	94.00	0
9.29b(5,4)		92.51	0.43	91.67	0		95.68	0.48	92.93	0
10.34b(6,4)	**92.29**	92.05	0.34	91.02	8.331	**96.30**	95.65	0.41	93.65	7.678
11.23b(7,4)	**92.13**	91.97	0.24	91.11	0	**96.32**	95.68	0.28	93.33	0
12.00b(8,4)	**92.29**	92.03	0.12	91.78	0	**95.97**	94.70	0.13	93.24	0

While QAT follows the warm-up phase gradually stabilizes training and reduces loss. Figure 2 shows that warm-up delays convergence and introduces additional training overhead, without yielding clear accuracy gains. Consequently, although warm-up can partially alleviate PTQ instability, it does not offer a practical advantage, and training directly with low-bit lattice-based QAT from the outset is both simpler and more effective.

e) Overload behavior and effect of Δ_0 In hierarchical lattice quantization, Δ_0 controls the size of the coarse quantizer and thus directly governs overload. A vector overloads when it falls outside the Δ_0-scaled Voronoi cell of the coarse lattice, which triggers additional residual digits through $g_m(x) = q^{-m}\left(Q_L^{\circ m}(x) - Q_L^{\circ(m+1)}(x)\right)$, thereby requiring deeper traversal through the M-level hierarchy. Thus, overload is simply a signal that the hierarchy must use more digits to represent the vector; it is not an error condition.

When Δ_0 is small, the effective coarse cell becomes large, so fewer vectors fall outside it, and overload frequency decreases. However, a very small Δ_0 compresses the dynamic range, and the most significant digits may not be fully utilized. When Δ_0 is large, the coarse cell becomes tight, making it

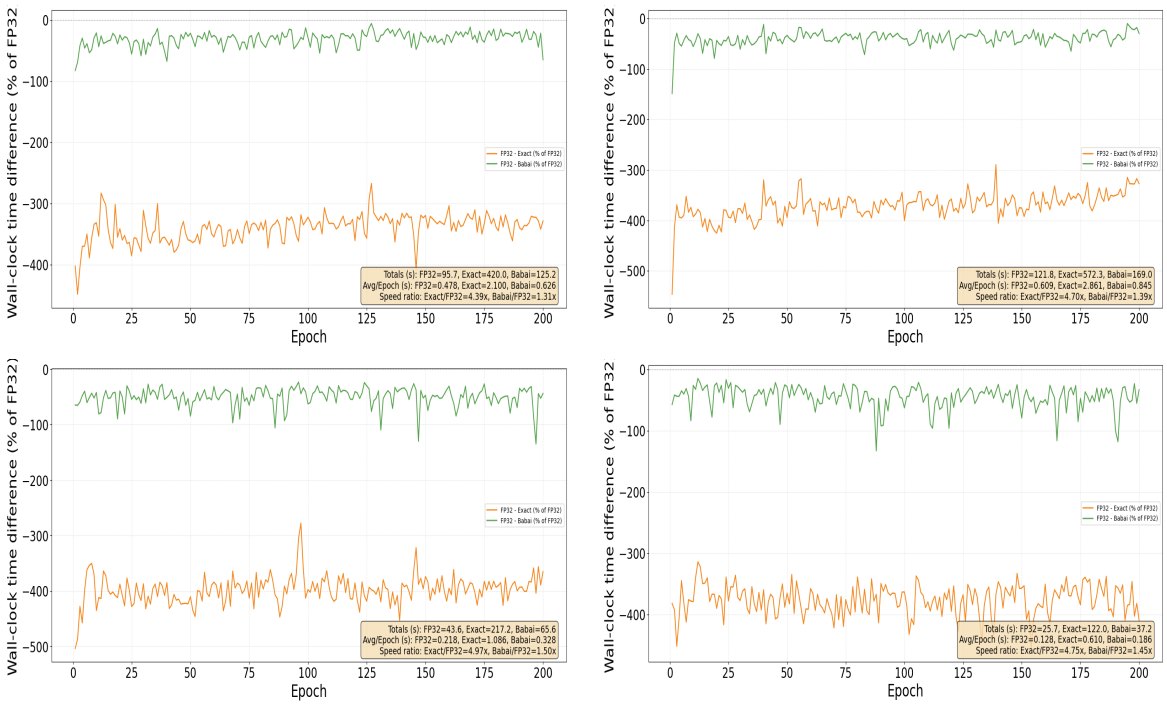

Figure 1: Comparison of quantization time of FP32 vs VQ for ReCon and AntShield in the first row and IMDB and SMS Spam in the second row, respectively.

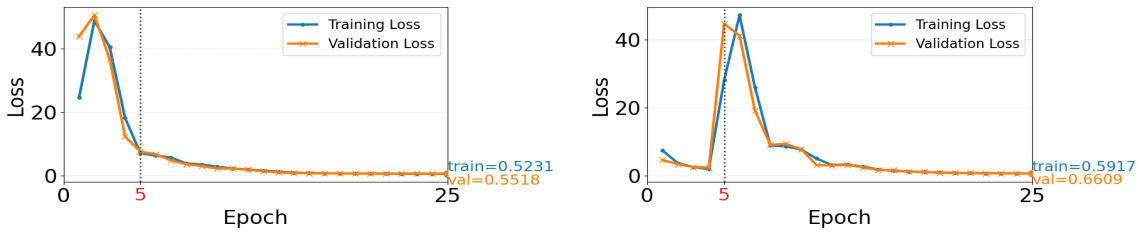

Figure 2: Effect of warm-up on the 3.00b(2,3) configuration for ReCon. Left: without warm-up; right: with warm-up.

more likely that vectors exceed it, and thus, overload frequency increases. In such regimes, one may need to increase q, M, or both to ensure that the representation can encode the required variation through deeper hierarchical digits.

If a vector exceeds the base cell only slightly, the resulting residual can still have a very small norm, meaning the quantization error at the base level remains low. In such cases typically when overload rates are $< 1\%$, one may deliberately omit residual correction to gain compute and time savings. This is a designer's choice: skipping residual refinement may slightly inflate measured accuracy but does not meaningfully harm fidelity when overload is negligible. Conversely, once overload $> 1\%$, the base projection alone is no longer faithful, and residual correction becomes essential. Configurations with high accuracy and low overload are ideal, whereas high accuracy and high overload remain acceptable but require residual correction, increasing compute cost. In contrast, regimes with low accuracy, regardless of overload, indicate unsuitable quantization settings.

Our experiments show that when overload is reduced either by choosing an appropriate Δ_0 or by selecting q, M pairs that avoid excessive residuals, the quantizer behaves more stably, and performance at low precision improves. Thus, reducing overload reliably mitigates degradation and restores stable behavior in the low-bit regime.

f) Comparing Δ_0: We study the sensitivity of lattice-based quantization to the choice of the base scaling parameter Δ_0 by comparing a smaller $\Delta_0 = 0.3$ against a higher setting of $\Delta_0 = 1.5$ across datasets and decoding strategies as shown in Figure 3.

Across all datasets, $\Delta_0 = 1.5$ yields consistently stable performance for both Exact and Babai, with F1 scores closely matching those achieved at moderate and high bit-widths. In contrast, $\Delta_0 = 0.3$ exhibits pronounced instability, particularly for Exact projection at low bit-widths. This is most evident on AntShield and SMS Spam datasets, where Exact with $\Delta_0 = 0.3$ collapses at 1-2 bits and only recovers once sufficient precision is available.

Babai is markedly more robust to aggressive scaling. Even with $\Delta_0 = 0.3$, Babai maintains competitive performance at low bit-widths. This robustness is consistent across datasets, indicating that approximate decoding mitigates the sensitivity to scaling mismatch.

As bit-width increases, the impact of Δ_0 diminishes. Beyond approximately 4-5 bits, both Δ_0 settings converge to similar performance levels, and differences between Exact and Babai decoding become marginal. This suggests that Δ_0 primarily governs behavior in the low-bit regime, where quantization noise and overload effects dominate. Overall, these results highlight a trade-off between aggressiveness and stability: a smaller Δ_0 can be effective when combined with robust decoding (Babai), but Exact decoding requires a more conservative scale to avoid overload.

g) QUIP# results

While OPTQ[32] (also referred to as GPTQ) is a widely used PTQ method — particularly within the Hugging Face ecosystem and LLM-focused workflows — we selected QuIP as a baseline because it is a more recent and strictly stronger method than OPTQ. As noted by the QuIP authors, GPTQ/OPTQ can be viewed as a special case of QuIP: in the absence of incoherence-inducing preprocessing, QuIP reduces to OPTQ. In this sense, the inclusion of QuIP already subsumes GPTQ-style approaches for the regimes considered in this paper. QuIP# [19] from the same authors improved QuIP by incorporating E8 lattice-based vector quantization codebooks, enabling more expressive and structured quantization. In this sense, QuIP#, with its incoherent preconditioning and lattice quantization, subsumes GPTQ-style approaches for our target settings and is a more relevant baseline for our setting.

For the aforementioned reasons, we additionally evaluate QuIP# for quantizing the BERT encoder and keep the MLP head quantized/trained using our existing E8-QAT setup. We run a two-pass procedure over the BERT's linear layers: (i) compute and cache a layer-wise Hessian using forward hooks on training batches, and (ii) quantize each layer using the cached Hessian and a chosen codebook. For efficiency and stability, we skip attention projection layers (query/key/value and attention output projection) and optionally exclude embeddings/LayerNorm. After replacing the corresponding BERT linear modules with quantized ones, we freeze all BERT parameters (PTQ mode) and train only the MLP parameters. We repeat this for multiple matched-precision settings by selecting the codebook and MLP configuration per bit-width (e.g., 2/3/4-bit: BERT codebook $\rightarrow$ MLP (q, M)). Table 6 shows that 2-bit PTQ benefits from Babai decoding in all datasets except AntShield, while 3–4 bits yield better performance in the Exact setting across datasets.

Table 6: QUIP# on BERT and QAT on MLP.

B-bits	M-Bits	VQ-Exact		VQ-Babai		VQ-Exact		VQ-Babai	
		F1*	OL%*	F1*	OL%*	F1*	OL%*	F1*	OL%*
		ReCon				**IMDB**			
2.00	2.00b(4,1)	56.53/85.29	0.016/0.04,0.02	**64.26**/67.32	0/0,0	44.58/78.54	0.008/0.04,0.01	**64.85**/66.67	0/0,0
3.00	3.00b(8,1)	60.48/87.65	0.016/0.06,0.01	58.55/68.02	0/0,0	**64.61**/87.12	0.018/0.06,0.02	62.76/78.33	0/0,0
4.00	4.00b(16,1)	**61.94**/NA	0.031/NA	64.08/NA	0/NA	51.96/NA	0.031/NA	55.16/NA	0/NA
		AntShield				**SMS Spam**			
2.00	2.00b(4,1)	47.30/89.17	0.014/0.03,0.01	37.10/57.06	0/0,0	63.44/96.05	0.014/0.03,0.01	73.00/7.55	0/0,0
3.00	3.00b(8,1)	**62.35**/90.95	0.012/0.06,0.01	56.18/27.85	0/0,0	84.21/98.65	0.012/0.05,0.01	**81.18**/26.57	0/0,0
4.00	4.00b(16,1)	60.63/NA	0.020/NA	**62.22**/NA	0/NA	**89.87**/NA	0.022/NA	75.26/NA	0/NA

* Comparing F1- score of QUIP# on BERT + QAT on MLP / BERT+MLP end-to-end QAT (Refer Table 3).
*Comparing OL% of QUIP# on BERT + QAT on MLP (MLP overload)/ BERT+MLP end-to-end QAT (BERT overload, MLP overload).

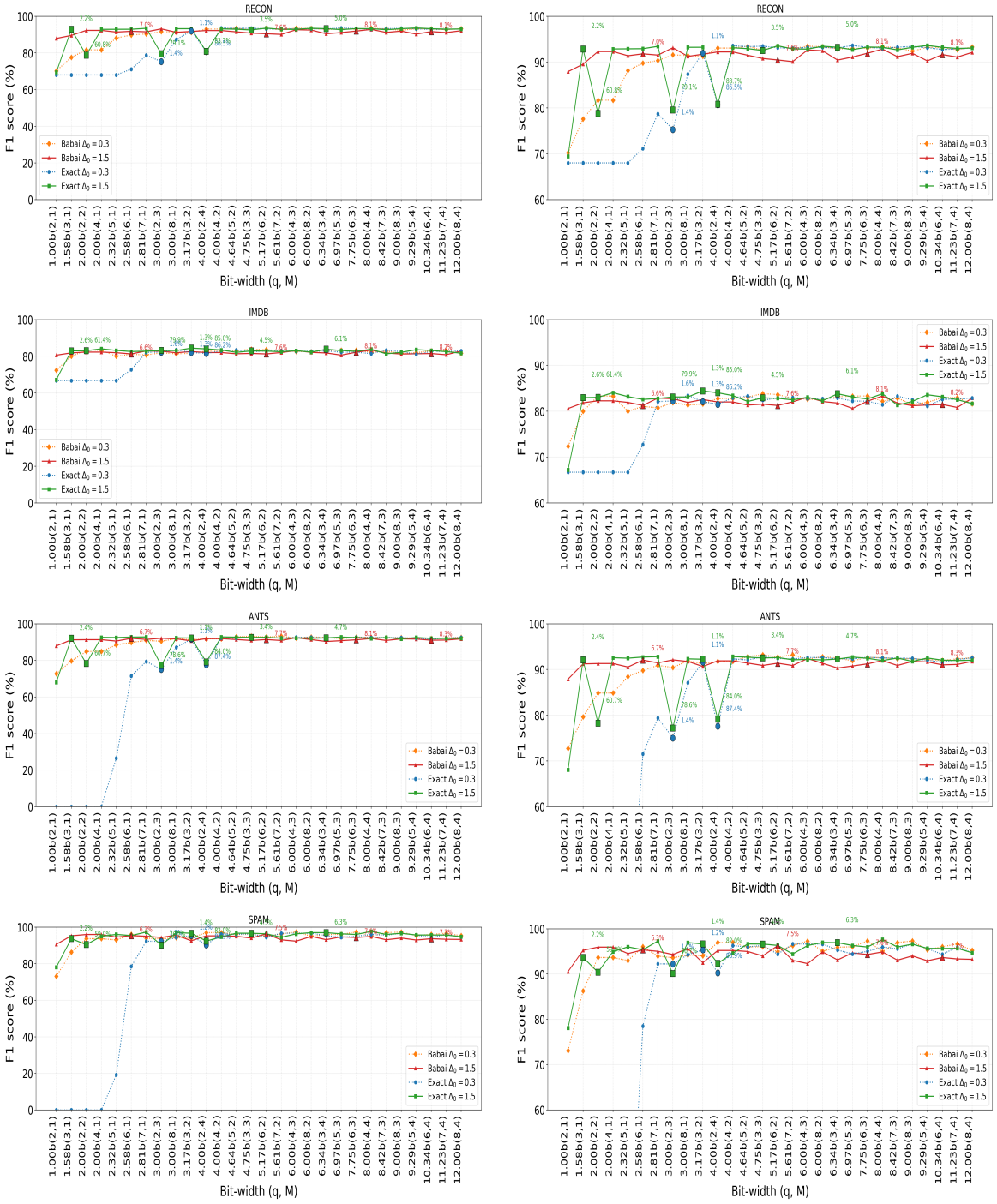

Figure 3: Comparison of VQ for Δ_0 (full and zoomed version of left and right, respectively).

We also apply QuIP# in a pure PTQ setting to the MLP classifier operating on fixed BERT embeddings. Specifically, we quantize only the first MLP layer (768→512) using QuIP#, while keeping activations and the classification head in FP32. The Hessian for the quantized layer is computed from training embeddings, cached, and reused across runs. This setting isolates the effect of QuIP# weight-only PTQ on downstream classification performance. In contrast to end-to-end quantization, weight-only PTQ of the MLP using QuIP# (Table 7) shows minimal accuracy degradation even at low precision, confirming that most sensitivity arises from encoder quantization rather than the classifier head.

Table 7: QuIP# on MLP after training FP32 BERT+MLP

M-Bit(s)	**ReCon***	**IMDB***	**AntShield***	**SMS Spam***
2.00	93.63/92.87, 92.30	85.26/84.05, 82.28	92.93/92.59, 91.31	**97.28**/94.95, 95.95
3.00	**93.99**/93.23, 91.25	85.70/83.12, 91.92	**93.44**/92.33, 92.11	96.95/96.95, 95.62
4.00	93.93/93.13, 92.22	**86.02**/84.03, 82.03	92.97/92.83, 91.90	96.95/94.59, 95.21

* Comparing F1- score of QUIP# on FP32 MLP trained on FP32 BERT embeddings / QAT (Exact, Babai)on MLP trained on FP32 BERT embeddings (Refer Table 2).

Overall, the results show that BERT+MLP end-to-end QAT beats PTQ on BERT and QAT on MLP. Whereas, in setting wherein only MLP is quantized and BERT remains full precision, QUIP# is comparable or better than QAT on MLP.

B. Exploring D_4 Lattice

To assess the generality of the proposed VQ framework beyond E_8 lattice, we additionally explore D_4 lattice. This allows us to verify that the core design is not specific to E_8, but extends naturally to other lattice families.

B.1. Nearest-neighbor projection of D_4 lattice

The D_4 lattice consists of integer vectors with even coordinate sum, enabling a simple parity-based correction rule. Algorithm 7 implements exact nearest-neighbor projection onto D_4 by first rounding to $\mathbb{Z}^4$ and, when the parity constraint is violated, flipping the coordinate with the largest rounding residual to obtain the closest valid lattice point.

Algorithm 7 Nearest-neighbor projection $Q_{D_4}(x)$

Input: $x \in \mathbb{R}^4$ (or batch $x \in \mathbb{R}^{B\times 4}$)
Output: $y \in D_4$

3 $f \leftarrow \mathrm{custom\_round}(x)$; $y \leftarrow \begin{cases} f, & \text{if } (\sum_i f_i) \bmod 2 = 0 \\ g(x), & \text{otherwise} \end{cases}$

4 **Helper-1:** $\mathrm{custom\_round}(v) = \lfloor v - \mathrm{sign}(v)\varepsilon + \frac{1}{2} \rfloor$, where $\varepsilon > 0$ is tiny (ties at ± 0.5 resolved toward 0).

5 **Helper-2:** Let $f(x) = \mathrm{custom\_round}(x)$ and $k = \arg\max_j |x_j - f(x)_j|$. Then $g(x)$ equals $f(x)$ except at coordinate k, which is changed by ± 1, so that the parity of $\sum_i f(x)_i$ flips and ties are resolved consistently.

B.2. Quantization of only MLP

Table 8 reports results for quantizing only the MLP classifier while keeping the BERT in FP32, comparing Exact VQ based on the E_8 and D_4 lattices. At 1.0 bits, both lattices exhibit lower F1 score than higher bid-widths, but still are stable. As the effective bit-rate increases to ≈1.58 bits and beyond, there is a significant lift in performance, maintaining negligible overload. These results indicate that the proposed VQ framework applies consistently across lattice choices in a setting where only MLP is quantized.

Table 8: VQ Exact results using D4 Lattice for FP32 BERT and QAT on MLP.

M-Bit(s)	F1*	OL%*	F1*	OL%*	F1*	OL%*	F1*	OL%*
	ReCon		IMDB		AntShield		SMS Spam	
1.00b(2,1)	69.46/70.58	0.02/0	67.22/67.41	0/0	68.04/66.77	0.04/0	78.10/49.03	0.01/0
1.58b(3,1)	92.87/88.31	2.25/6.96	82.97/82.80	2.62/7.27	92.09/86.64	2.38/7.53	93.65/94.39	2.25/13.31
2.00b(2,2)	78.79/93.60	60.81/0	83.03/83.79	61.42/0	78.25/92.65	60.68/0	90.41/95.59	59.01/0
2.00b(4,1)	92.87/93.37	0.02/0.04	84.05/83.04	0.02/0.03	92.59/92.85	0.01/0.02	94.95/96.27	0.01/0.03
2.32b(5,1)	92.90/93.08	0/0.46	83.19/83.23	0.01/0.56	92.48/92.03	0/0.48	96.00/97.30	0.01/0.63
2.58b(6,1)	92.93/93.20	0.01/0.01	82.59/82.58	0.01/0.09	92.73/92.59	0.01/0.02	95.36/96.95	0.02/0.04
2.81b(7,1)	93.44/93.30	0/0.09	82.79/81.76	0.01/0.13	92.82/92.75	0/0.10	97.28/95.92	0.01/0.19
3.00b(2,3)	79.56/93.26	79.14/0	83.12/84.18	79.88/0	77.21/92.89	78.64/0	90.10/96.97	76.70/0
3.00b(8,1)	93.23/93.26	0/0.01	83.12/82.77	0/0.01	92.33/92.34	0.01/0	96.95/95.33	0.01/0.04
3.17b(3,2)	93.24/93.25	1.00/4.54	84.43/83.34	1.28/4.89	92.25/92.53	1.14/4.45	96.64/92.11	1.42/4.84
4.00b(2,4)	80.82/93.11	83.71/0	84.03/84.44	84.97/0	79.12/92.66	84.03/0	92.36/96.35	81.96/0
4.00b(4,2)	93.13/93.48	0.40/0.02	83.42/82.39	0.39/0.04	92.83/92.96	0.35/0.01	94.59/95.39	0.30/0.08

* Comparing F1- score and overload of QAT (Exact) on MLP trained on FP32 BERT embeddings (**E8/D4**) (Refer Table 2 for E8).

Table 9: VQ Exact results using D4 Lattice for BERT(B)+MLP(M) end-to-end QAT.

Config	ReCon		IMDB		AntShield		SMS Spam	
	F1*	OL%(B,M)*	F1*	OL%(B,M)*	F1*	OL%(B,M)*	F1*	OL%(B,M)*
1.00b(2,1)	68.25/78.56	0.41,0.27/0,0	82.37/81.56	0.44,0.27/0,0	71.33/81.41	0.41,0.28/0,0	0/95.36	0.48,0/0.27,0
2.00b(4,1)	85.29/86.43	0.04,0.02/0.05,0.03	78.54/84.09	0.04,0.01/0.04,0.02	89.17/87.27	0.03,0.01/0.04,0.02	96.05/95.42	0.03,0.01/0.05,0.02
3.00b(8,1)	87.65/87.57	0.06,0.01/0.07,0.04	87.12/85.42	0.06,0.02/0.08,0.03	90.95/90.47	0.06,0.01/0.07,0.03	98.65/97.33	0.05,0.01/0.08,0.03

* Comparing F1- score of QAT (Exact) BERT+MLP trained end-to-end (**E8/D4**) (Refer Table 2 for E8).
* Comparing overload (Bert overload, MLP overload) of QAT (Exact) BERT+MLP trained end-to-end (**E8/D4**) (Refer Table 3 for E8)

B.3. Quantization of both BERT and MLP

Table 9 reports end-to-end QAT results when both the BERT and MLP classifier are quantized using Exact VQ, for D_4 lattice (comparing to E_8). At 1 bit, model is unstable for SMS Spam dataset while gives descent performance at such low precision for other datasets, accompanied by minimal overload in both BERT and MLP layers.

As the weight precision increases to 2–3 bits, accuracy improves substantially, with both lattices yielding comparable trends. Overall, the results indicate that the proposed VQ framework extends naturally to end-to-end quantization of both BERT and MLP across different lattices.

C. Activation quantization

While the main paper focuses on weight-only quantization (activations kept in FP32), we additionally study whether activation quantization can be combined with the proposed weight VQ. We evaluate QAT for weight quantization with ACIQ-based[27] activation in two settings: (i) quantizing only the MLP and keeping BERT in full precision, and (ii) end-to-end BERT+MLP QAT.

In our implementation, each quantized linear layer applies ACIQ after the linear output. Running activation statistics are tracked using EMA, and a symmetric clipping threshold α is computed analytically. We use the mixed ACIQ variant, which selects between Laplace and Gaussian-based thresholds by minimizing estimated MSE. Activations are uniformly quantized with step size $\Delta = \alpha/2^{b_a-1}$.

C.1. Quantization of only MLP

Across datasets, the results in Table 10 suggest the presence of multiple operating regimes depending on the relative precision of weights and activations. Configurations with higher weight precision and higher activation precision (e.g., 3-bit weight with 8-bit activation) tend to exhibit comparatively stable performance across datasets, with lower overload. In contrast, configurations with similar weight precision (e.g., 3-bit) but reduced activation precision ($<$2 bits) mostly show bit-reduced performance in ReCon and AntShield datasets.

When both weights and activations are quantized aggressively ($<$2 weight and $<$ 8 activation bits), training becomes unstable. In contrast, maintaining high activation precision while moderately

quantizing weights appears to be a safe choice. These observations indicate that, for a fixed weight precision (e.g., $\leq$ 3 bits), higher activation precision (8 bits) represents a safer operating region, while reducing activation bits introduces a more challenging regime that requires careful handling.

Table 10: Results of FP32 BERT and MLP-QAT using E8 VQ-Exact with fixed clipping=2.0 on weights and for activations using ACIQ. Here, 'W' means weight bit and 'A' means activation bit. FP-A means full precision activation (without ACIQ).

M-Bits(W/A)	ReCon			IMDB			AntShield			SMS Spam		
	FP-A	F1	OL%	FP-A	F1	OL%	FP-A	F1	OL%	FP-A	F1	OL%
3.00(2,3)/2	79.56	70.42	1.82	83.12	75.66	2.11	77.21	72.89	2.10	90.10	76.64	2.13
3.00(2,3)/3		73.25	1.51		77.76	1.63		72.23	1.65		70.24	2.00
3.00(2,3)/4		75.31	1.32		77.61	1.60		71.30	1.38		72.51	1.83
3.00(2,3)/8		73.25	1.51		77.76	1.63		72.23	1.65		70.24	2.00
3.00(8,1)/2	93.23	80.13	0.00	83.12	78.67	0.00	92.33	80.39	0.00	96.95	91.33	0.00
3.00(8,1)/3		83.73	0.00		80.31	0.00		83.98	0.00		92.11	0.00
3.00(8,1)/4		84.28	0.00		79.11	0.00		86.34	0.00		95.62	0.00
3.00(8,1)/8		83.73	0.00		80.31	0.00		83.98	0.00		92.11	0.00
3.17(3,2)/2	93.24	83.74	1.67	84.43	79.34	1.40	92.25	84.41	1.78	96.64	92.47	1.29
3.17(3,2)/3		86.61	1.51		80.00	1.49		86.68	1.66		95.02	1.32
3.17(3,2)/4		87.08	1.44		80.08	1.38		88.06	1.61		94.63	1.30
3.17(3,2)/8		86.61	1.51		80.00	1.49		86.68	1.66		95.02	1.32
4.00(2,4)/2	80.82	76.17	82.75	84.03	71.83	79.42	79.12	75.30	83.80	92.36	66.09	78.74
4.00(2,4)/3		79.10	88.09		73.03	85.09		80.02	89.57		71.38	83.88
4.00(2,4)/4		75.71	89.99		75.96	86.22		79.79	91.14		77.22	84.89
4.00(2,4)/8		75.71	89.99		75.96	86.22		79.79	91.14		77.22	84.89
4.00(4,2)/2	93.13	89.04	0.00	83.42	79.19	0.00	92.83	88.76	0.00	94.59	92.93	0.00
4.00(4,2)/3		90.23	0.00		81.13	0.00		90.16	0.00		92.31	0.00
4.00(4,2)/4		90.53	0.00		80.48	0.00		89.65	0.00		93.92	0.00
4.00(4,2)/8		90.53	0.00		80.48	0.00		89.65	0.00		93.92	0.00

Table 11: Results of end-to-end QAT of both BERT and MLP using E8 VQ with fixed clipping=2.0 on weights and for activations using ACIQ. Here, 'W' means weight bit and 'A' means activation bit. FP-A(QE/QB) means full precision activation (without ACIQ) in end-to-end Exact and Babai settings, respectively (referring to Table 3).

Config(W/A)	FP-A (QE/QB)	VQ-E	VQ-B	FP-A (QE/QB)	VQ-E	VQ-B	FP-A (QE/QB)	VQ-E	VQ-B	FP-A (QE/QB)	VQ-E	VQ-B
	ReCon			IMDB			AntShield			SMS Spam		
2.00(4,1)/2	85.29/67.32	52.79	53.56	78.54/66.67	51.82	52.88	89.17/57.06	45.86	45.00	96.05/7.55	96.55	16.35
2.00(4,1)/3		**71.92**	36.19		82.35	52.80		82.27	33.38		**97.99**	12.27
2.00(4,1)/4		58.60	53.83		1.18	**62.42**		**84.78**	35.48		97.33	**25.27**
2.00(4,1)/8		70.80	**56.09**		**85.07**	1.19		76.11	**57.15**		96.05	2.33
3.00(8,1)/2	87.65/68.02	70.43	58.50	87.12/78.33	50.67	58.06	90.95/27.85	59.20	46.70	98.65/26.57	97.99	20.85
3.00(8,1)/3		85.07	38.43		84.57	65.36		88.61	40.00		97.99	23.53
3.00(8,1)/4		**87.14**	**64.48**		85.77	66.42		**91.11**	39.91		**99.33**	**24.24**
3.00(8,1)/8		86.94	64.44		**86.03**	**66.67**		91.04	**67.32**		97.96	23.93

C.2. Quantization of both BERT and MLP

Table 11 reports results for end-to-end QAT. Similar to the MLP-only setting, the results indicate distinct operating regions depending on the joint precision of weights and activations. Configurations with moderate-to-high weight precision (e.g., 3 bits) and higher activation precision (4–8 bits) tend to show more stable behavior across datasets, with performance closer to the FP-activation reference.

When activation precision is reduced while keeping weights fixed (e.g., 3-bit weights with $\leq$3-bit activations), performance becomes more variable, and the gap relative to FP-A widens. More aggressive joint quantization of both weights and activations (e.g., 2-bit activations combined with lower-weight precision) exhibits larger degradation and instability, particularly in the end-to-end setting. Overall, the results suggest that, for end-to-end BERT+MLP QAT, operating with moderately quantized weights and comparatively higher activation precision represents a safer regime, whereas lowering activation precision introduces a more challenging region that requires careful tuning.

Overall, these results indicate that incorporating activation quantization alongside weight quantization can be effective, achieving performance close to the corresponding weight-only configura-

tions with full-precision activations (*FP-A* column in Table 10 and 11). Therefore, the proposed framework naturally extends to joint quantization of both weights and activations within a model. However, when activation quantization is extreme, we need to switch to higher weight bits.

D. Experiments on multi-class classification dataset

We additionally evaluate the proposed quantization framework on **AG's News**, a four-class news topic classification benchmark, to assess behavior beyond binary classification. The model setup follows the same pipeline, with experiments covering PTQ and QAT, Exact and Babai settings, both SQ and VQ schemes, and D_4 and E_8 lattices.

Table 12: PTQ vs. QAT ($\Delta_0 = 1.5$ for all VQ settings; $\Delta_0 = 0.3$ for Babai PTQ). E and B denote Exact and Babai, respectively.

M-Bit(s)	QAT			PTQ			
	Brevitas	VQ-E	VQ-B	Brevitas	VQ-E	VQ-B	QuIP
	AG'S News						
1.00	Error	53.39	**88.59**	Error	16.59	23.02	86.13
1.58	-	**90.76**	89.68	-	84.24	32.99	-
2.00	90.69	**90.87**	90.54	74.26	87.12	43.72	89.25
2.32	-	**91.13**	89.26	-	88.13	54.55	-
2.58	-	**90.62**	90.45	-	88.45	67.32	-
2.81	-	**91.08**	89.84	-	88.51	69.06	-
3.00	90.62	**91.06**	90.55	89.28	86.97	74.14	90.91
3.17	-	**90.96**	89.76	-	88.26	78.86	-
4.00	91.20	91.09	90.94	90.52	87.66	85.25	**91.29**

PTQ vs. QAT. As seen in Table 12, different regimes emerge across bit-widths on the multi-class dataset. At 1 bit, VQ with Babai under QAT achieves strong performance, whereas PTQ methods largely fail or severely degrade. At 1.58 bits, most QAT variants (VQ-Exact and VQ-Babai) converge to comparable accuracy, whereas PTQ with Babai remains unstable. Beyond 1.58 bits, QAT with VQ-Exact consistently delivers the best performance, while PTQ, particularly Babai, continues to exhibit instability and under-performance below 3 bits. These results highlight the advantage of QAT over PTQ in low-precision multi-class settings.

Table 13: Comparison of SQ and VQ with Δ_0=1.5. Here, OL represents overload.

M-Bits(s)	SQ	VQ-Exact		VQ-Babai	
		F1	OL%	F1	OL%
	AG's-News (FP: 91.25)				
1.00b(2,1)	Error	53.39	0.01	88.59	0
1.58b(3,1)		**90.76**	2.59	89.68	0
2.00b(2,2)	89.46	83.96	58.00	90.54	0
2.00b(4,1)		90.87	0.02	90.54	0
2.32b(5,1)		**91.13**	0	89.26	0
2.58b(6,1)		90.62	0	90.45	7.58
2.81b(7,1)		91.08	0	89.84	0
3.00b(2,3)	88.90	84.25	73.90	90.55	0
3.00b(8,1)		90.96	0.47	89.76	0
3.17b(3,2)		**91.06**	0	90.34	0
4.00b(2,4)	88.80	86.18	82.67	90.94	0
4.00b(4,2)		**91.09**	0.20	90.94	0

SQ vs. VQ Table 13 shows that at 1 bit, SQ fails to produce a valid model, whereas VQ with Babai achieves strong performance, highlighting the advantage of lattice structure in the 1-bit regime. For bit-widths $\geq$1.58, all methods begin to recover accuracy; however, VQ, particularly VQ-Exact, consistently matches or outperforms SQ at the same effective bit-rate, while maintaining negligible overload. Overall, these results indicate that VQ provides a more robust and scalable alternative to SQ, especially at low and intermediate precision levels.

End-to-end BERT quantization. Table 14 reports end-to-end BERT+MLP quantization results under PTQ and QAT. Across all bit-widths, PTQ-based approaches (both Exact and Babai) exhibit the weakest performance. At 1 bit, QAT with VQ-Exact remains stable when activations are kept in full precision, whereas introducing aggressive activation quantization leads to instability. Higher activation precision (8 bits) consistently yields safer and more reliable behavior across configurations. At > 8 bits it should remain the same.

Table 14: Comparison of BERT FP32 and quantization performance. Here, ^{PE}:PTQ with Exact, ^{PB}:PTQ with Babai, ^{QE}:QAT with Exact, ^{QB}:QAT with Babai, A: ACIQ on activations with (bits), ^{FP}: FP32. For Exact Δ_0=1.5 and for babai Δ_0=0.3

Config	FP32	$B^{QE}M^{QE}$	$B^{QE}M^{QE}$-A	$B^{F}M^{PE}$	$B^{F}M^{PB}$	$B^{F}M^{QE}$	$B^{F}M^{QB}$
	AG News						
1.00b(2,1)		90.64	-	16.59	23.02	53.39	88.59
2.00b(4,1)	91.25	91.66	92.61(8)	87.12	42.24	90.87	90.54
3.00b(8,1)		93.48	93.16(8)	86.97	73.33	91.06	90.34

Table 15: VQ Exact results using D4 Lattice for FP32 BERT and QAT on MLP.

M-Bit(s)	F1*	OL%*
1.00b(2,1)	53.39/83.99	0.014/0.000
1.58b(3,1)	90.76/87.58	2.588/6.230
2.00b(2,2)	83.96/91.09	57.998/0.000
2.00b(4,1)	90.87/91.20	0.020/0.022
2.32b(5,1)	91.13/90.60	0.002/0.196
2.58b(6,1)	90.62/91.08	0.008/0.000
2.81b(7,1)	91.08/91.30	0.004/0.005
3.00b(2,3)	84.25/91.29	73.903/0.000
3.00b(8,1)	91.06/90.86	0.000/0.000
3.17b(3,2)	90.96/91.00	0.466/2.206
4.00b(2,4)	86.18/90.89	82.674/0.000
4.00b(4,2)	91.09/90.57	0.199/0.000
4.00b(16,1)	91.20/91.01	0.004/0.000

* Comparing F1- score and overload of QAT (Exact) on MLP trained on FP32 BERT embeddings (**E8/D4**).

Using D_4 Lattice. To assess whether the proposed VQ framework generalizes beyond the E_8 lattice, we additionally evaluate a D_4-based VQ-Exact variant. Table 15 and 16 show results of using D_4 lattice in VQ-Exact setup for MLP-only QAT and end-to-end BERT+MLP QAT, respectively. Across bit-widths, the performance and overload behavior obtained with D_4 are comparable to those achieved with E_8, with similar accuracy trends and low overload in stable operating regimes. These observations indicate that the proposed quantization framework is not tied to a specific lattice choice and can be extended to alternative lattices.

Overall, the AG's News experiments confirm that the observed trends on binary datasets extend to multi-class classification: VQ offers improved robustness over SQ, QAT mitigates low-bit degradation, and Exact setup remains more stable than Babai at lower precision.

Table 16: VQ Exact results using D4 Lattice for BERT(B)+MLP(M) end-to-end QAT.

Config	AG News	
	F1	OL%(B,M)
1.00(2,1)	90.64/90.79	0.62, 0.23/0,0
2.00(4,1)	91.66/90.89	0.03, 0.01/0.04,0.03
3.00(8,1)	93.48/90.78	0.05, 0.01/0.08,0.04

* Comparing F1- score of QAT (Exact) BERT+MLP trained end-to-end (**E8/D4**).
* Comparing overload (Bert overload, MLP overload) of QAT (Exact) BERT+MLP trained end-to-end (**E8/D4**).