Trainable Bitwise Soft Quantization for Input Feature Compression

Karsten Schrödter[1], Jan Stenkamp[1], Nina Herrmann[1], Fabian Gieseke[1,2]
[1]University of Münster (Germany), [2]University of Copenhagen (Denmark)
[first_name].[last_name]@uni-muenster.de
fabian.gieseke@di.ku.dk

The growing demand for machine learning applications in the context of the Internet of Things calls for new approaches to optimize the use of limited compute and memory resources. Despite significant progress that has been made w.r.t. reducing model sizes and improving efficiency, many applications still require remote servers to provide the required resources. However, such approaches rely on transmitting data from edge devices to remote servers, which may not always be feasible due to bandwidth, latency, or energy constraints. We propose a task-specific, trainable feature quantization layer that compresses the input features of a neural network. This can significantly reduce the amount of data that needs to be transferred from the device to a remote server. In particular, the layer allows each input feature to be quantized to a user-defined number of bits, enabling a simple on-device compression at the time of data collection. The layer is designed to approximate step functions with sigmoids, enabling trainable quantization thresholds. By concatenating outputs from multiple sigmoids, introduced as bitwise soft quantization, it achieves trainable quantized values when integrated with a neural network. We compare our method to full-precision inference as well as to several quantization baselines. Experiments show that our approach outperforms standard quantization methods, while maintaining accuracy levels close to those of full-precision models. In particular, depending on the dataset, compression factors of $5\times$ to $16\times$ can be achieved compared to 32-bit input without significant performance loss.

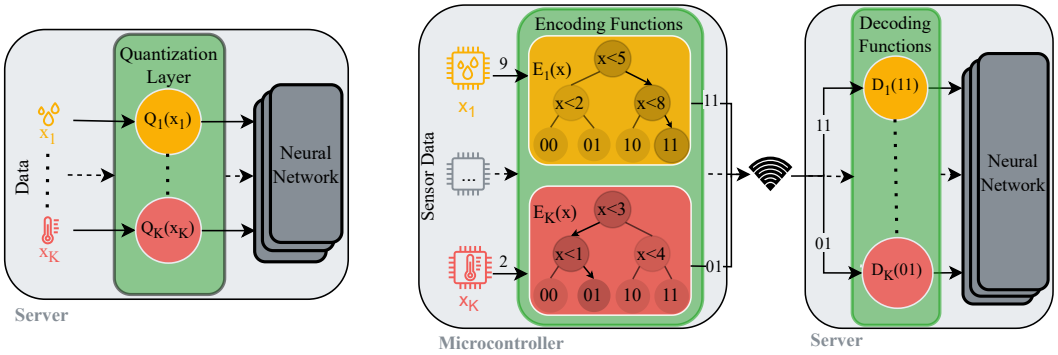
1. Introduction

The number of Internet of Things (IoT) applications has steadily grown in recent years, with use cases spanning a wide range of domains [1–4]. These applications are often based on microcontrollers that are equipped with sensors that measure parameters such as temperature, humidity, pressure, or vibrations. Typically, such microcontrollers have very limited compute and memory capabilities (e.g., only 2 KB of RAM), which severely restricts the types of algorithms that can be executed "locally" on the devices. This necessitates the development of efficient, resource-aware implementations tailored to the specific needs of the given application and hardware.

Several approaches have been proposed for deploying machine learning models in resource-constrained settings. One common strategy is to reduce the model size to an extent that inference can be performed on the device. Nevertheless, this approach requires trade-offs between model complexity and the available resources [5, 6]. Alternatively, the collected sensor data can also be transmitted to more powerful remote machines for further processing. However, this strategy also poses significant challenges. Reliable communication links and sufficient energy supplies might not be available. In extreme cases, devices are battery-powered and can only transmit very limited amounts of data—sometimes just a few bytes per hour. This is the case, for instance, with communication protocols such as LoRaWAN. Transmitting data via mobile networks (e.g., LTE) is in contrast often energy-intensive and might also not be possible due to a lack of cellular coverage [7].

We focus on scenarios where (a) on-device execution of a machine learning model is not feasible, and (b) only very limited amounts of data can be transmitted to a remote server. Such situations

Third Conference on Parsimony and Learning (CPAL 2026).



(a) Training (b) Inference

Figure 1: The trainable feature quantization layer is integrated into a neural network to learn task-specific compressions for each input feature: (a) During training, the quantization layer (green rectangle) and the neural network are trained jointly on a remote server. The layer gives rise to an encoder-decoder composition $\mathrm{Q}_i = \mathrm{D}_i \circ \mathrm{E}_i$ for the i-th input feature. (b) During inference, the encoder E_i is used to encode the i-th feature on the resource-constrained device using lightweight coding logic. The compressed features are then sent to the server, where the decoder D_i is used to decode the i-th feature. All decoded features are then used as input for the remaining neural network, which is executed on the remote server.

commonly arise when IoT devices are deployed in remote locations, e.g., for environmental monitoring (forestry, wildlife, agriculture, ...). Hence, the devices mainly collect data, and strategies are required to minimize the size of data transmitted. Straightforward approaches are based on selecting important features (e.g., via forward feature selection [8]). Another strategy involves compressing the collected (sensor) data by using lower precision for floating-point numbers. However, naive precision reduction is task-agnostic and often degrades downstream model performance.

We propose an end-to-end trainable feature quantization layer that can be integrated into a (deep) neural network to learn task-specific compressions for each input feature. The resulting encoding scheme is based on simple intervals and can be efficiently implemented using standard *if-then-else* rules. As shown in Figure 1, the quantization layer is jointly trained with a given neural network on a remote server to determine optimal thresholds and quantized values for efficient deployment. During inference, quantization is implemented on a resource-constrained device using lightweight logic, while decoding is performed on a more capable server, where the remaining neural network processes the compressed input. By integrating quantization into training, the method adapts to the underlying data distribution, enabling flexible and effective splitting of input features.

2. Background

We start by summarizing related work and our contributions.

2.1. Related Work

Several approaches have been proposed to execute machine learning methods on IoT nodes. The main challenge in this context is to reduce both computational and memory demands to enable execution on highly resource-constrained hardware—without significant loss in model accuracy. This has led to the *tiny machine learning* paradigm [5, 6, 9, 10]. When sufficiently optimized, such "tiny" models can run directly on microcontrollers, enabling compact and intelligent devices. However, reducing the model size generally leads to inferior results, especially when the IoT devices are extremely resource-constrained. As a result, offloading the collected data to more powerful remote machines still remains a common approach for many applications. Data transfer during the inference phase is also an active area of research. Broadly, existing approaches fall into two categories:

(a) feature extraction, which transforms the original features into a new representation, and (b) feature selection, which identifies a relevant subset of the original features (e.g., [11–16]).

The research area of *collaborative intelligence* [17] explores the possibility of distributing inference tasks across multiple devices, with each device being responsible for computing a portion of the entire model. Kang et al. [18] first proposed a collaborative intelligence approach, called *Neurosurgeon*, which automatically determines split points for deep neural networks in a layer-wise manner. Works based on this approach also consider reducing the size of the data to be transmitted by focusing on image compression [19–22]. Another topic of interest is to actually find suitable split points for neural networks [23] and to identify so-called early-exit points [24].

Quantization is used to reduce the bit width of activations and weights of a neural network, see Gholami et al. [25] or Li et al. [26] for an overview. When it comes to training and inference, the two main approaches are based on applying quantization only during inference (post-training quantization) or on incorporating quantization into the training process (quantization-aware training). Research on post-training quantization includes, for example, output reconstruction [27, 28] and clustering of activations [29]. In quantization-aware training, non-differentiable functions are classically approximated by identity functions using the Straight-Through Estimator (STE) [30]. Further research approximates non-differentiable quantization layers with random noise [31] or applies differentiable soft quantization functions using a tanh function [32]. Furthermore, learnable quantizers have been investigated such as quantization with trainable step size [33] and learnable lookup tables [34]. Yang et al. [35] propose quantization networks and introduce soft quantization using sigmoid functions. Although this quantization enables flexible quantizer learning, empirical results show that learned thresholds generally do not outperform predefined ones.

2.2. Contribution

We focus on efficient quantization methods for input features. Building on the concept of soft quantization [35], which performs layer-wise compression of activations and weights using predefined quantized values, we propose two modifications: first, we transform this approach into feature-wise compression of inputs, allowing for learnable thresholds per feature; second, we incorporate bitwise quantization to facilitate the learning of quantized values. We also present a comprehensive evaluation of input data quantization across six datasets. Overall, our framework enables efficient feature quantization on low-power devices, requiring only a few if-else statements for encoding.

3. Methodology

A *quantization function* $\mathrm{Q} : \mathbb{R} \to \{v_0, \dots, v_M\}$ is based on $M+1$ quantized values for some predefined $M \in \mathbb{N}$. In this work, we consider scalars or binary vectors, i.e., $v_m \in \mathbb{R}$ or $v_m \in \{0,1\}^M$ for $0 \leq m \leq M$. The function Q can be split into an encoding function $\mathrm{E} : \mathbb{R} \to \{0, \dots, M\}$ and a decoding function $\mathrm{D} : \{0, \dots, M\} \to \{v_0, \dots, v_M\}$, defined by $m \mapsto v_m$, so that $\mathrm{Q} = \mathrm{D} \circ \mathrm{E}$.[1] In practice, a bit width of n-bits for some given $n \in \mathbb{N}$ is achieved by using $M = 2^n - 1$.

In *split inference* approaches, the data is first encoded using E, resulting in n-bit values that are sent to a remote server. There, the values are decoded using D and processed further. This process is illustrated in Figure 1b.

3.1. Encoding Function via Thresholds

We introduce encoding functions that are based on multiple thresholds by combining step-functions via summation. For a threshold $a \in \mathbb{R}$, the hard step-function $\mathbb{I}_{\geq a} : \mathbb{R} \to \{0,1\}$ is defined as

$$\mathbb{I}_{\geq a}(x) = \begin{cases} 1 & \text{if } x \geq a \\ 0 & \text{if } x < a \end{cases} .$$

[1] Note that Q and E are often both called quantization functions. Since we may set the decoding function D to the identity, each encoding function is also a quantization function.

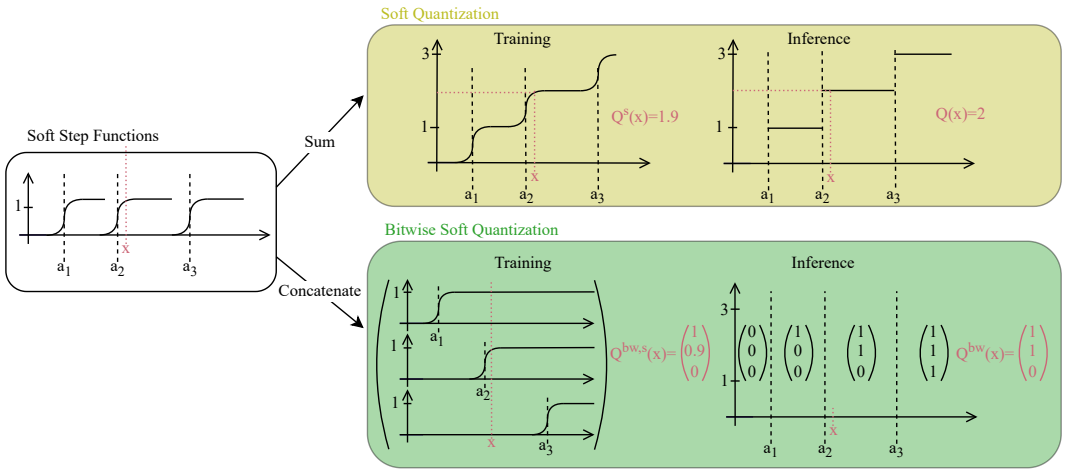


Figure 2: Schematic overview: Soft Quantization (Q^{s}) and Bitwise Soft Quantization ($\mathrm{Q}^{\mathrm{bw,s}}$) are formed by summing or concatenating multiple soft step functions. Both are differentiable with respect to thresholds, enabling optimization during training. During inference, they are converted into Hard Quantization (Q) and Bitwise Quantization (Q^{bw}) via rounding.

Multiple step-functions can be combined in order to define an encoding function based on thresholds as follows: consider M thresholds $a_1 < \cdots < a_M \in \mathbb{R}$. Then, the associated hard encoding function $\mathrm{E}_{a_1,\ldots,a_M} : \mathbb{R} \to \{0, 1, \ldots, M\}$ can be defined as

$$x \mapsto \sum_{m=1}^{M} \mathbb{I}_{\geq a_m}(x) \in \{0, \ldots, M\}. \tag{1}$$

3.2. Decoding Functions

In order to define a suitable decoding function, quantized values need to be defined. The trivial approach is to define the decoding function as the identity, so that $v_m = m$ for all $0 \leq m \leq M$, see Hard Quantization in Figure 2 for an example. We present two more ideas.

Another option is to resort to quantized values in the middle of the interval between two thresholds. For $M > 1$ and $0 \leq m \leq M$ we define

$$v_m \coloneqq \frac{1}{2} \cdot (a_m + a_{m+1}). \tag{2}$$

We need to define two auxiliary thresholds $a_0 \coloneqq 2 \cdot a_1 - a_2$ and $a_{M+1} \coloneqq 2 \cdot a_M - a_{M-1}$. The decoding function is then given by $\mathrm{D}_{a_1,\ldots,a_M} : \{0, \ldots, M\} \to \{v_0, \ldots, v_M\}$ with $m \mapsto v_m$. Finally, the quantization function associated with the thresholds $a_1 < \cdots < a_M \in \mathbb{R}$ is then given by

$$\mathrm{Q}_{a_1,\ldots,a_M} = \mathrm{D}_{a_1,\ldots,a_M} \circ \mathrm{E}_{a_1,\ldots,a_M}\,.$$

For instance, the *minmax quantization* divides the range of data points into equally sized intervals, where quantized values are given by the middle of the intervals, see, e.g., Gholami et al. [25, p. 5]. The idea of *quantile quantization* is to define the thresholds using quantiles, such that all quantized values occur equally often. Using quantile thresholds in the quantization function defined above leads to a slight modification of quantile quantization as described by Dettmers et al. [36, App. F.2]. We refer to Figure 3 for a visualization and to Appendix A for a detailed description of both minmax and quantile quantization.

3.2.1. Bitwise Quantization via Thresholds

Another option to define decoding functions is to consider binary vectors as quantized values. The underlying idea is to have a quantization function that concatenates step-functions, instead of sum-

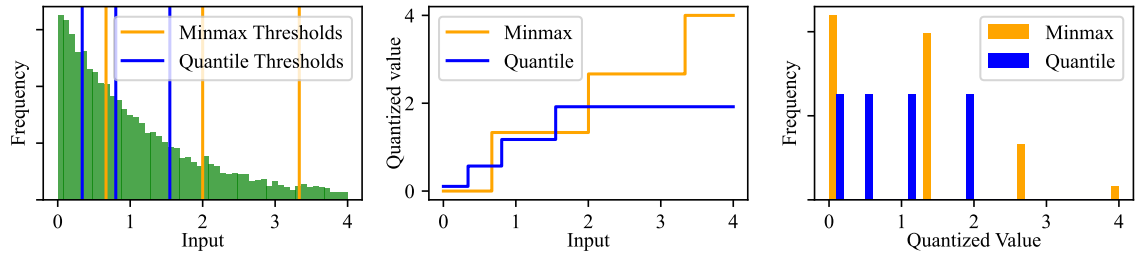


Figure 3: Example of a quantization with bit width $n = 2$ (i.e., $M = 3$ thresholds) using minmax and quantile quantization on artificial skewed data. Left: Histogram of input data together with thresholds for minmax and quantile quantization. Center: Minmax and quantile quantization functions. Right: Histogram of quantized values for minmax and quantile quantization.

ming them. For $0 \leq m \leq M$, we define the bitwise quantized value v_m^{bw} as follows:

$$v_m^{\text{bw}} := [\underbrace{1, 1, \ldots, 1}_{m \text{ ones}}, \underbrace{0, 0, \ldots, 0}_{M-m \text{ zeros}}]^{\mathsf{T}} \in \{0, 1\}^M. \tag{3}$$

The decoding function $\mathrm{D}^{\text{bw}} : \{0, \ldots, M\} \to \{0, 1\}^M$ with $m \mapsto v_m^{\text{bw}}$ is the bitwise decoding function and the composition $\mathrm{Q}^{\text{bw}}_{a_1,\ldots,a_M} = \mathrm{D}^{\text{bw}} \circ \mathrm{E}_{a_1,\ldots,a_M} : \mathbb{R} \to \{0, 1\}^M$ is called bitwise quantization, where $a_1 < \cdots < a_M \in \mathbb{R}$ are M thresholds and $\mathrm{E}_{a_1,\ldots,a_M}$ is the encoding function defined above. For $x \in \mathbb{R}$ the bitwise quantization is given by

$$\mathrm{Q}^{\text{bw}}_{a_1,\ldots,a_M}(x) = [\mathbb{I}_{\geq a_1}(x), \ldots, \mathbb{I}_{\geq a_M}(x)]^T \in \{0, 1\}^M. \tag{4}$$

See Bitwise Soft Quantization in Figure 2 for an example.

Using bitwise quantized values as input for a neural network can be helpful, as composing $\mathrm{Q}^{\text{bw}}_{a_1,\ldots,a_M}$ with a linear layer $h : \mathbb{R}^M \to \mathbb{R}$ yields a quantization function $h \circ \mathrm{Q}^{\text{bw}}_{a_1,\ldots,a_M} : \mathbb{R} \to \{\tilde{v}_0, \ldots, \tilde{v}_M\}$ with learnable quantized values. We refer to Appendix B for mathematical details.

In the context of multiple inputs, a linear layer that receives bitwise quantized values as input and produces a single neuron can be viewed as mapping each bitwise-quantized feature to a learned quantized scalar and then summing these scalars to produce the output. A similar layer with multiple outputs is therefore able to process different quantized values per input feature. Thus, when applying bitwise quantization to each input feature and integrating it with a neural network, the first layer of the network effectively learns and combines quantized values in diverse ways tailored to the specific task. This approach can be beneficial, as optimal quantized values may vary depending on the task.

3.3. Approximating with Soft Quantization

This section introduces soft quantization, which enables the training of thresholds using gradient-based methods. Soft quantization functions, introduced by Yang et al., are approximations of hard quantization functions, where the step-functions are replaced by sigmoidal functions to avoid vanishing gradients. Building on this, we also introduce bitwise soft quantization.

Soft Step Function

Soft step functions are translated and stretched sigmoid functions. For $a \in \mathbb{R}$ and a temperature parameter $\tau > 0$ the soft step function $\mathbb{I}^{\text{s}}_{\geq a} : \mathbb{R} \to (0, 1)$ is

$$\mathbb{I}^{\text{s}}_{\geq a}(x) = \frac{1}{1 + \exp\left(\frac{a-x}{\tau}\right)} = \sigma\left(\frac{x-a}{\tau}\right), \tag{5}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the sigmoid function. Note that the threshold a is used to translate the sigmoid function and τ is a stretching factor, inspired by the temperature parameter in distilling [37]. Due

to the shape of the sigmoid function, when rounding the output of the soft step function, the step function is reproduced, i.e.

$$\mathbb{I}_{\geq a_m}(x) = \text{round}\left(\mathbb{I}^{\text{s}}_{\geq a_m}(x)\right) \ \forall x \in \mathbb{R}.$$

Soft Encoding and Quantization Function

As with hard quantization functions, soft quantization functions are built by combining multiple soft step functions either through summation (soft quantization) or concatenation (bitwise soft quantization). As above, consider M thresholds $a_1 < \cdots < a_M \in \mathbb{R}$ and additionally a temperature parameter $\tau > 0$. Define the associated soft quantization function $\mathrm{Q}^{\text{s}}_{a_1,\ldots,a_M} : \mathbb{R} \to (0, M)$ and bitwise soft quantization function $\mathrm{Q}^{\text{bw,s}}_{a_1,\ldots,a_M} : \mathbb{R} \to (0,1)^M$ by

$$\mathrm{Q}^{\text{s}}_{a_1,\ldots,a_M}(x) = \sum_{i=1}^{M} \mathbb{I}^{\text{s}}_{\geq a_m}(x) \in (0, M) \tag{6}$$

$$\mathrm{Q}^{\text{bw,s}}_{a_1,\ldots,a_M}(x) = [\mathbb{I}^{\text{s}}_{\geq a_1}(x), \ldots, \mathbb{I}^{\text{s}}_{\geq a_M}(x)]^T \in (0,1)^M \tag{7}$$

for $x \in \mathbb{R}$. See (Bitwise) Soft Quantization in Figure 2 for an example. The notation for soft quantization slightly differs from the one in Yang et al. [35], but the resulting functions have the same shape. The approximation of the (bitwise) hard quantization function by the (bitwise) soft quantization function improves with decreasing $\tau > 0$. Note that (bitwise) soft quantization is differentiable with respect to all thresholds a_m. For more details on the derivative, we refer to Appendix C. The (bitwise) soft quantization function can be used to learn thresholds using gradient-based methods.

3.4. Quantization Layer

A quantization layer quantizes each input feature independently and concatenates the quantized values into one output vector. We refer to Appendix D for a mathematical description. If all quantization functions are soft quantization functions, the layer is called a soft quantization layer. Included at any step of a deep neural network, a soft quantization layer can be used in order to learn the best quantization thresholds. Note that during inference, all soft quantization functions are converted to hard quantization functions to ensure a smaller bit width for the encoded values. Therefore, it makes sense to decrease τ during the training process in order to improve the approximation of a hard quantization layer and decrease the effect of rounding.

4. Experiments

We compare the performance of soft bitwise quantization with that of several baseline quantization methods on a number of regression datasets. Additionally, we conduct an ablation study to differentiate between soft and bitwise quantization. To demonstrate the minimal overhead caused by our encoding approach, we conducted additional experiments on the latency and energy requirements of an implementation on a microcontroller. The respective results are included in Appendix E. [2]

4.1. Experimental Setup

The following sections describe the evaluated datasets, models, and training process.

4.1.1. Datasets

As features are represented with a varying number of bits, our approach is tested on regression tasks using 6 datasets that have continuous features with a variety of values. The prediction tasks of the datasets cover a wide range; from housing prices (dataset name California Housing [38]) or relative compute time of cpu in user mode (CPU Activity Database[3]), sinusoidal formula reproduction (Fried [39, 40]), hydrogen sulfide concentration (Sulfur [41]), temperature of superconductors

[2] The code for all experiments is available at `https://github.com/kschr40/FeatureCompression`

[3] `https://www.openml.org/search?type=data&status=active&id=197`

Name	Abbreviation	Source
Full Precision	FP	Baseline
Pre Minmax Quantization	Pr-MQ	Baseline
Pre Quantile Quantization	Pr-QQ	Baseline
Learnable Step Size Quantization	LSQ	[33]
Learnable Lookup Table with granule 4	LLT4	[34]
Learnable Lookup Table with granule 9	LLT9	[34]
Bitwise Soft Quantization	Bw-SQ	Ours

Table 1: Models

(Superconduct [42]) and quality of wine (Wine Quality [43]). The number of instances varies from about 6500 in Wine Quality to more than 40000 in Fried, while the datasets have between 7 (Sulfur) and 80 (Superconduct) features. The Wine Quality dataset also has on average the fewest distinct values per feature, about 241. In contrast, the other datasets contain on average more than 1000 distinct values per feature, up to 8878 for California Housing. We refer to Appendix F, Table 5 for more information about the datasets.

4.1.2. Models

We propose to use a Bitwise Soft Quantization (Bw-SQ) model for joint training of quantization and the subsequent neural network. The thresholds for the bitwise soft quantization layer are initialized using the thresholds of quantile quantization.

In order to benchmark our model, five approachess are analyzed as baselines, see Table 1 for an overview of used models. The first model is a plain multi-layer perceptron (MLP) using bit width 32 without any quantization, called the full precision (FP) model. Two additional baselines are minmax and quantile quantization, applied to the input features during training and inference, simulating a quantization-aware training approach for input feature compression. Furthermore, we use two more recent approaches, described in more detail below.

Learned Step Size Quantization [33] (LSQ) This approach approximates the gradient of the quantization function by using a straight-through estimator [30], which effectively leads to a learnable step size (or scale parameter) in quantization. Due to lack of an official github repository, we use an unofficial one[4] and include the code in our training pipeline.

Learnable Lookup Tables [34] (LLT) This work formulates quantization as a lookup table, made differentiable by approximating a hard decision using a softmax operation. In consequence, learnable lookup tables start by creating too many thresholds (by a factor called granule) and learns to keep the best thresholds. In the original work, a granule of 9 was suggested. In contrast, preliminary experiments suggested that granule 4 works well for our use cases. Therefore, we are comparing against both variants in our experiments.

4.1.3. Model Training and Selection

In order to compare the performance of our models with the baseline models, hyperparameter tuning is performed for all methods. This is done to ensure a fair comparison between all methods, which might perform best using different hyperparameter settings. The basic architecture for all models (except the FP model) contains a quantization layer and a subsequent MLP. The quantization layer compresses each feature to the desired bit width. In our experiments, the bit width varies from $n = 2$ bits to $n = 8$ bits per feature. The MLP is a dense neural network, where the number of hidden layers and neurons per hidden layer are hyperparameters. All models are trained using mean squared error (MSE) loss together with dropout. The dropout rate, the learning rate, and the number of epochs to train are also hyperparameters. An additional hyperparameter is given by the decrease factor of the temperature parameter τ for soft quantization layers. τ is initialized as

[4] https://github.com/zhutmost/lsq-net

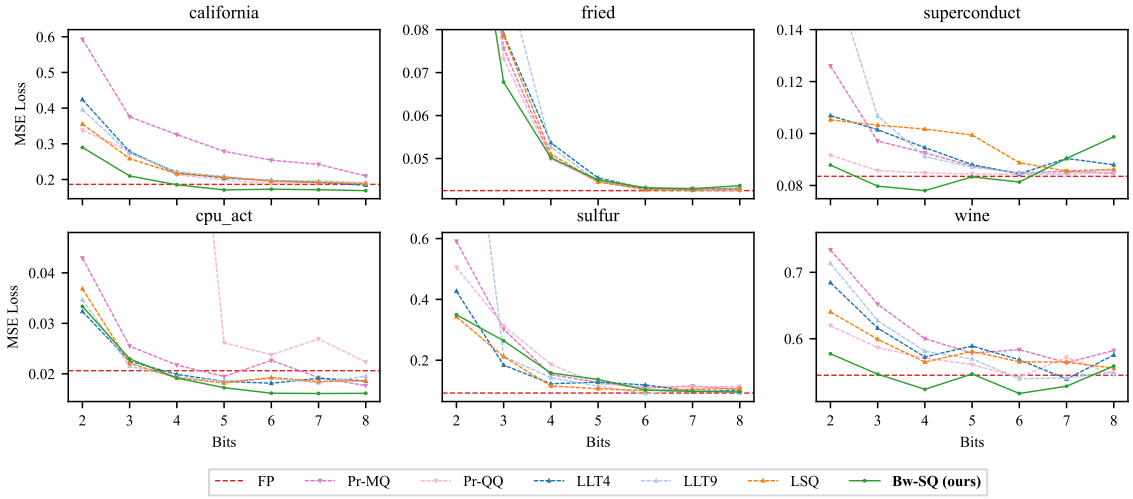


Figure 4: MSE values per dataset per quantization level of the best-performing hyperparameter setting averaged over 10 data splits for all methods. The red line is the average over the full precision measurements.

1 and exponentially decreased after each epoch, such that it equals the decrease factor at the end of training, see also Appendix G for justification of this choice. During preprocessing, all features and labels are standardized. 4-fold cross-validation is used to determine the best hyperparameter setup per method, bit width and dataset. Finally, the best hyperparameter setting is applied with 10 different train-test splits (90% train, 10% test data) to get a robust estimate of the generalization performance. In addition to the average MSE of the best performing hyperparameter setup, a 95%-confidence interval for the average MSE across all 10 data splits is calculated. The results of two methods are called significantly different if the confidence intervals are disjoint. For details on hyperparameter tuning, see Appendix H.

4.2. Results

The main results of our experiments are visualized in Figure 4, showing the average test loss of the trained MLPs for the different bit widths. The corresponding table of results, as well as a table of 95%-confidence intervals, are contained in Appendix J. With a few exceptions, the loss is decreasing for all quantization methods with increasing bit width across the evaluated datasets, converging to each other and towards the performance of the FP model. In over half of the experiments (26 out of 42), Bw-SQ yields the smallest average MSE loss across all quantization methods.

Comparison with Quantization Techniques. For the datasets Wine Quality, California Housing, CPU Activity Database and Fried, it can be seen that Bw-SQ outperforms the comparison quantization methods, partly by far. On Superconduct, Bw-SQ also performs well for small bit widths, but the error increases for higher bit widths, which might indicate numerical instability on this dataset. Finally, Bw-SQ only performs well on Sulfur dataset for bit widths 2 and 6 to 8 and is outperformed by e.g. LSQ for small bit widths up to 6. In particular, there is no quantization technique that performs best for all datasets. For the comparison methods, it can be seen that in general LLT works better with the smaller granule 4 for small bit width. Between LLT and LSQ, the superior model is dataset-dependent. Another observation is that comparing pre-defined thresholds, Pr-QQ performs better on 3 datasets (Wine Quality, California Housing and Superconduct), Pr-MQ performs better on 2 datasets (CPU Activity Database and Sulfur), while the performance is similar on Fried. Therefore, quantile thresholds do not always outperform minmax thresholds. Nevertheless, Bw-SQ – initialized with quantile thresholds – is still able to perform well on CPU Activity Database due to learnable thresholds.

Comparison With Full Precision Model. We investigate the bit width tipping point, defined as the minimal bit width, such that there is no significant difference between Bw-SQ and FP. This varies between the datasets. For Superconduct and Wine Quality, there is no significant difference for a bit width of $n = 2$, having a small performance drop of 5% from FP to Bw-SQ for Superconduct and 6% for Wine Quality. For California Housing and CPU Activity Database, the tipping point is at bit width $n = 3$ and Bw-SQ even performs better than FP for bit widths up from 4 bits. On the datasets Sulfur, the tipping point for bit width equals $n = 4$, although FP (MSE 0.093) performs much better in average than Bw-SQ (MSE 0.158). On the artificial dataset Fried, the tipping point is at bit width $n = 6$. Overall, Bw-SQ is able to compress data by an average compression factor of $11.1\times$ with a range from $5\times$ to $16\times$ without a significant performance loss compared to the FP model with bit width 32. For some configurations, Bw-SQ outperforms the FP model, for example on CPU Activity Database for bit widths up from 4 bits. This may be due to a regularization effect arising from their reduced representational capacity, which can prevent overfitting of the model on the dataset.

4.3. Ablation Study

To test which component of Bitwise Soft Quantization is responsible for the performance gain and to what extent, we perform an ablation study. We evaluate three variants of our model: The first model performs Soft Quantization (SQ), i.e. it has trainable thresholds, while the second and third model use Bitwise Minmax Quantization (Bw-MQ), resp. Bitwise Quantile Quantization (Bw-QQ). In Table 2 we collect the relative performance to Bitwise Soft Quantization, averaged over the datasets. More detailed results are presented in Appendix J, Table 11. It can be seen that none of the ablation methods can robustly perform similarly to Bw-SQ. For example, Bw-QQ slighly outperforms Bw-SQ on Wine Quality and Superconduct, but does not perform good on Sulfur and CPU Activity Database. In average, SQ is the best of the ablation methods. Nevertheless, it performs approximately 12% worse than Bw-SQ. In conclusion, it can be seen that Bw-SQ is able to join the advantage of learnable thresholds from SQ and of the bitwise decoding as in Bw-QQ.

Dataset	california	fried	superconduct	cpu_act	sulfur	wine	Mean
Bw-SQ	0.195	0.061	0.086	0.020	0.172	0.543	0.180
SQ	+12.15%	+3.74%	+5.86%	+28.34%	+13.36%	+7.36%	+11.80%
Bw-MQ	+57.43%	+8.03%	+16.42%	+5.74%	+21.21%	+4.86%	+18.95%
Bw-QQ	+9.91%	+6.64%	−0.75%	+112.85%	+33.49%	−1.36%	+26.80%

Table 2: Average MSE of Bitwise Soft Quantization per dataset together with average MSE ratio of ablation methods against Bitwise Soft Quantization per dataset. Values below 0 indicate better performance than Bitwise Soft Quantization.

5. Conclusion

In this work, we introduce Bitwise Soft Quantization as a framework to combine learnable thresholds from soft quantization [35] with learnable quantized values from bitwise quantization. This leads to a quantization technique that can be used to compress input features of neural networks by a factor $5\times$ to $16\times$ without significant performance loss compared to the 32 bit full precision model. Results indicate that the regularization effect of quantization can even lead to better generalization on some datasets. However, this work is limited by its focus solely on MLP models and regression data, as well as the equal bit width compression applied to all features. Addressing these limitations will be the focus of future research.

Acknowledgements

This work was funded by the German Federal Ministry for the Environment, Nature Conservation, Nuclear Safety and Consumer protection, Project TinyAIoT, Funding Nr. 67KI32002A. We also acknowledge the computational resources provided by the PALMA II cluster at the University of Münster (subsidized by the DFG; INST 211/667-1).

References

[1] Syed Mujtaba Hassan Rizvi, Asma Naseer, Shafiq Ur Rehman, Sheeraz Akram, and Volker Gruhn. Revolutionizing Agriculture: Machine and Deep Learning Solutions for Enhanced Crop Quality and Weed Control. *IEEE Access*, 12:11865–11878, 2024. ISSN 2169-3536. doi: 10.1109/ACCESS.2024.3355017. URL `https://ieeexplore.ieee.org/document/10401912/`.

[2] Silvia Liberata Ullo and G. R. Sinha. Advances in smart environment monitoring systems using iot and sensors. *Sensors*, 20(11), 2020. ISSN 1424-8220. doi: 10.3390/s20113113. URL `https://www.mdpi.com/1424-8220/20/11/3113`.

[3] Abbas Shah Syed, Daniel Sierra-Sosa, Anup Kumar, and Adel Elmaghraby. Iot in smart cities: A survey of technologies, practices and challenges. *Smart Cities*, 4(2):429–475, 2021. ISSN 2624-6511. doi: 10.3390/smartcities4020024. URL `https://www.mdpi.com/2624-6511/4/2/24`.

[4] Mostafa Haghi Kashani, Mona Madanipour, Mohammad Nikravan, Parvaneh Asghari, and Ebrahim Mahdipour. A systematic review of iot in healthcare: Applications, techniques, and trends. *Journal of Network and Computer Applications*, 192:103164, 2021. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2021.103164. URL `https://www.sciencedirect.com/science/article/pii/S1084804521001764`.

[5] Pete Warden and Daniel Situnayake. *TinyML: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, Inc., 2019.

[6] Vijay Janapa Reddi, Brian Plancher, Susan Kennedy, Laurence Moroney, Pete Warden, Anant Agarwal, Colby R. Banbury, Massimo Banzi, Matthew Bennett, Benjamin Brown, Sharad Chitlangia, Radhika Ghosal, Sarah Grafman, Rupert Jaeger, Srivatsan Krishnan, Maximilian Lam, Daniel Leiker, Cara Mann, Mark Mazumder, Dominic Pajak, Dhilan Ramaprasad, J. Evan Smith, Matthew Stewart, and Dustin Tingley. Widening access to applied machine learning with tinyml. *CoRR*, abs/2106.04008, 2021. URL `https://arxiv.org/abs/2106.04008`.

[7] Višnja Križanović, Krešimir Grgić, Josip Spišić, and Drago Žagar. An Advanced Energy-Efficient Environmental Monitoring in Precision Agriculture Using LoRa-Based Wireless Sensor Networks. *Sensors*, 23(14):6332, January 2023. ISSN 1424-8220. doi: 10.3390/s23146332. URL `https://www.mdpi.com/1424-8220/23/14/6332`. Number: 14 Publisher: Multidisciplinary Digital Publishing Institute.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[9] Albert Gural and Boris Murmann. Memory-optimal direct convolutions for maximizing classification accuracy in embedded applications. In *Internatinal Conference on Machine Learning* (*ICML*), pages 2515–2524, 2019.

[10] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 kb ram for the internet of things. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1935–1944. PMLR, August 2017. URL `https://proceedings.mlr.press/v70/kumar17a.html`.

[11] Muhammed Fatih Balın, Abubakar Abid, and James Zou. Concrete autoencoders: Differentiable feature selection and reconstruction. In *ICML*, pages 444–453. PMLR, 2019.

[12] Feng Nan, Joseph Wang, and Venkatesh Saligrama. Pruning random forests for prediction on a budget. In *NeurIPS*, pages 2334–2342. Curran Associates, 2016.

[13] Stefan Oehmcke and Fabian Gieseke. Input selection for bandwidth-limited neural network inference. In Arindam Banerjee, Zhi-Hua Zhou, Evangelos E. Papalexakis, and Matteo Riondato, editors, *Proceedings of the 2022 SIAM International Conference on Data Mining* (*SDM*), pages 280–288, USA, 2022. SIAM Publications. doi: 10.1137/1.9781611977172.32.

[14] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *ICLR*. OpenReview.net, 2017.

[15] Xiaowei Xu et al. Scaling for edge inference of deep neural networks. *Nature Electr.*, 1(4): 216–222, 2018.

[16] Ismael Lemhadri, Feng Ruan, Louis Abraham, and Robert Tibshirani. Lassonet: A neural network with feature sparsity. *JMLR*, 22(127):1–29, 2021.

[17] Ivan V. Bajić, Weisi Lin, and Yonghong Tian. Collaborative intelligence: Challenges and opportunities. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing* (*ICASSP*), pages 8493–8497, 2021. doi: 10.1109/ICASSP39728.2021.9413943.

[18] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, April 2017. ISSN 0163-5964. doi: 10.1145/3093337.3037698.

[19] Yoshitomo Matsubara, Davide Callegaro, Sameer Singh, Marco Levorato, and Francesco Restuccia. Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing. In *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks* (*WoWMoM*), pages 337–346. IEEE, June 2022. doi: 10.1109/wowmom54355.2022.00032.

[20] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. Supervised compression for resource-constrained edge computing systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (*WACV*), pages 2685–2695, January 2022. URL `https://openaccess.thecvf.com/content/WACV2022/html/Matsubara_Supervised_Compression_for_Resource-Constrained_Edge_Computing_Systems_WACV_2022_paper.html`.

[21] Saurabh Singh, Sami Abu-El-Haija, Nick Johnston, Jona Ballé, Abhinav Shrivastava, and George Toderici. End-to-end learning of compressible features. In *2020 IEEE International Conference on Image Processing* (*ICIP*), pages 3349–3353. IEEE, October 2020. doi: 10.1109/icip40778.2020.9190860.

[22] Zhongzheng Yuan, Samyak Rawlekar, Siddharth Garg, Elza Erkip, and Yao Wang. Feature compression for rate constrained object detection on the edge. In *2022 IEEE 5th International Conference on Multimedia Information Processing and Retrieval* (*MIPR*), pages 1–6. IEEE, August 2022. doi: 10.1109/mipr54900.2022.00008.

[23] Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. Auto-split: A general framework of collaborative edge-cloud ai. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 2543–2553. ACM, August 2021. doi: 10.1145/3447548.3467078.

[24] Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, pages 1–15. ACM, September 2020. doi: 10.1145/3372224.3419194.

[25] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference, 2021. URL `https://arxiv.org/abs/2103.13630`.

[26] Min Li, Zihao Huang, Lin Chen, Junxing Ren, Miao Jiang, Fengfa Li, Jitao Fu, and Chenghua Gao. Contemporary advances in neural network quantization: A survey. In *2024 International Joint Conference on Neural Networks* (*IJCNN*), pages 1–10. IEEE, June 2024. doi: 10.1109/ijcnn60899.2024.10650109.

[27] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving Neural Network Quantization without Retraining using Outlier Channel Splitting. *International Conference on Machine Learning* (*ICML*), pages 7543–7552, June 2019.

[28] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? Adaptive rounding for post-training quantization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7197–7206. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/nagel20a.html`.

[29] Robert A. Cohen, Hyomin Choi, and Ivan V. Bajić. Lightweight compression of neural network feature tensors for collaborative intelligence. In *2020 IEEE International Conference on Multimedia and Expo* (*ICME*), pages 1–6, July 2020. doi: 10.1109/ICME46284.2020.9102797. URL `http://arxiv.org/abs/2105.06002`. arXiv:2105.06002 [cs].

[30] Yoshua Bengio. Estimating or propagating gradients through stochastic neurons, 2013. URL `https://arxiv.org/abs/1305.2982`.

[31] Saeed Ranjbar Alvar and Ivan V. Bajić. Multi-task learning with compressible features for collaborative intelligence. In *2019 IEEE International Conference on Image Processing* (*ICIP*), pages 1705–1709, 2019. doi: 10.1109/ICIP.2019.8803110.

[32] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (*ICCV*), October 2019. URL `https://openaccess.thecvf.com/content_ICCV_2019/html/Gong_Differentiable_Soft_Quantization_Bridging_Full-Precision_and_Low-Bit_Neural_Networks_ICCV_2019_paper.html`.

[33] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=rkgO66VKDS`.

[34] Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Suzanne Petryk, Sarah Adel Bargal, and Joseph E. Gonzalez. NBDT: neural-backed decision tree. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[35] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*). IEEE, June 2019. doi: 10.1109/cvpr.2019.00748.

[36] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization, 2022. URL `https://arxiv.org/abs/2110.02861`.

[37] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

[38] R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

[39] Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, 19(1): 1–67, 1991.

[40] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[41] Luigi Fortuna, Salvatore Graziani, Alessandro Rizzo, Maria G Xibilia, et al. *Soft sensors for monitoring and control of industrial processes*, volume 22. Springer, 2007.

[42] Kam Hamidieh. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*, 154:346–354, 2018. ISSN 0927-0256. doi: https://doi.org/10.1016/j.commatsci.2018.07.052. URL https://www.sciencedirect.com/science/article/pii/S0927025618304877.

[43] Paulo Cortez, Juliana Teixeira, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Using data mining for wine quality assessment. In João Gama, Vítor Santos Costa, Alípio Mário Jorge, and Pavel B. Brazdil, editors, *Discovery Science*, pages 66–79, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04747-3.

[44] Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Mueller, Joaquin Vanschoren, and Frank Hutter. Openml-python: an extensible python api for openml. *arXiv*, 1911.02490, 2020. URL https://arxiv.org/pdf/1911.02490.pdf.

A. Minmax and Quantile Quantization

A.1. Minmax Quantization

The *minmax quantization* divides the range of data points into equally sized intervals, where quantized values are given by the middle of the intervals, see, e.g., Gholami et al. [25, p. 5]. It can be recovered from our notation of quantization as follows:

For a given dataset of length $N \in \mathbb{N}$ consider all values of a selected continuous feature by $x_1, \ldots, x_N \in \mathbb{R}$. Define the minimal and maximal value as $x_{min} := \min\{x_1, \ldots, x_N\}$ and $x_{max} := \max\{x_1, \ldots, x_N\}$. Let $n \in \mathbb{N}$ be the targeted bit width, then define $M := 2^n - 1$ and the scale parameter s by

$$s := \frac{x_{max} - x_{min}}{M}.$$

Setting the thresholds $a_1 < \cdots < a_M \in \mathbb{R}$ to

$$a_m := x_{min} + (m - \frac{1}{2}) \cdot s, \; 1 \leq m \leq M$$

leads to quantization values (compare Equation 2)

$$v_m = x_{min} + m \cdot s, \; 0 \leq m \leq M.$$

The resulting quantization function can be reformulated as follows:

For $0 \leq m \leq M - 1$ consider some $x \in [a_m, a_{m+1})$. Then the following chain of inequalities hold true

$$x_{min} + (m - \frac{1}{2}) \cdot s = a_m \leq x < a_{m+1} = x_{min} + (m + \frac{1}{2}) \cdot s.$$

By subtracting x_{min} and dividing by s, this is equivalent to

$$m - \frac{1}{2} \leq \frac{x - x_{min}}{s} < m + \frac{1}{2}.$$

Thus, we may deduce that

$$x \in [a_m, a_{m+1}) \Leftrightarrow m = \text{round}(\frac{x - x_{min}}{s}).$$

Therefore, the associated quantization function $\mathrm{Q}_{a_1,\ldots,a_M} : \mathbb{R} \to \{v_1, \ldots, v_M\} \subset \mathbb{R}$ is given by

$$\mathrm{Q}_{a_1,\ldots,a_M}(x) = x_{min} + s \cdot \text{round}\left(\frac{x - x_{min}}{s}\right)$$

and thus is a slight modification of the quantization in Gholami et al. [25, p. 5] using the minimum and maximum of the signal as clipping range. Therefore, we will refer to this quantization function as minmax quantization. Please note that minmax quantization is sensitive to outliers as it works with minimal and maximal values.

A.2. Quantile Quantization

The idea of quantile quantization is to define the thresholds using quantiles, such that all quantized values occur equally often.

For a given dataset of length $N \in \mathbb{N}$ consider all values of a selected continuous feature by $x_1, \ldots, x_N \in \mathbb{R}$. For $\tau \in [0, 1]$ denote the τ-quantile of $x_1, \ldots, x_N$ by $q_\tau(x_1, \ldots, x_N)$. Let $n \in \mathbb{N}$ be the targeted bit width, then define $M := 2^n - 1$ and set

$$a_m := q_{\tau_m}(x_1, \ldots, x_N) \text{ with } \tau_m = \frac{m}{M+1}, \; 1 \leq m \leq M.$$

Using this, the quantized values $v_1, \ldots, v_M$ are given by Equation 2. The associated quantization function $\mathrm{Q}_{a_1,\ldots,a_M} : \mathbb{R} \to \{v_1, \ldots, v_M\}$ will be called the quantile quantization. Note that this is a slight modification of quantile quantization as in Dettmers et al. [36, App. F.2]. Please note that quantile quantization is less sensitive to outliers, as it works with quantiles instead of minmal and maximal values of a distribution.

B. Trainable Quantized Values via Bitwise Quantization

As in Section *Bitwise Quantization via Thresholds*, consider the bitwise quantization function $\mathrm{Q}^{\mathrm{bw}}_{a_1,\dots,a_M} : \mathbb{R} \to \{0,1\}^M$ and let $h : \mathbb{R}^M \to \mathbb{R}$ be a linear layer with M input dimensions and 1 output dimension, i.e., $h(z) = w^\mathsf{T} \cdot z + b$ for $z \in \mathbb{R}^M$ and some $w \in \mathbb{R}^M$ and $b \in \mathbb{R}$. Then, for $x \in [a_m, a_{m+1})$, we have (compare Equation 3):

$$\begin{aligned} \tilde{v}_m &\coloneqq h \circ \mathrm{Q}^{\mathrm{bw}}_{a_1,\dots,a_M}(x) = h(v^{\mathrm{bw}}_m) = w^\mathsf{T} \cdot v^{\mathrm{bw}}_m + b \\ &= \sum_{j=1}^{m} w_j + b. \end{aligned} \tag{8}$$

Thus, $h \circ \mathrm{Q}^{\mathrm{bw}}_{a_1,\dots,a_M} : \mathbb{R} \to \{\tilde{v}_0, \dots, \tilde{v}_M\}$ is itself a quantization function with the same thresholds $a_1 < \dots < a_M$, but with learnable quantized values $\tilde{v}_m$ for $0 \le m \le M$.

Note that one option would be to set the linear layer by $b \coloneqq v_0$ and $w = [w_1, \dots, w_M]^T$ with $w_m \coloneqq v_m - v_{m-1}$ for $1 \le m \le M$. Using Equation 8, it follows

$$\tilde{v}_m = \sum_{j=1}^{m} w_j + b = \sum_{j=1}^{m} v_m - v_{m-1} + v_0 = v_m$$

for all $0 \le m \le M$. Therefore, the quantization function $\mathrm{Q}_{a_1,\dots,a_M}$ is an example of $h \circ \mathrm{Q}^{\mathrm{bw}}_{a_1,\dots,a_M}$. Nevertheless, the composition $h \circ \mathrm{Q}^{\mathrm{bw}}_{a_1,\dots,a_M}$ is much more flexible.

C. Derivative of Soft Quantization Functions

The partial derivative of the soft quantization function with respect to the thresholds is given by

$$\begin{aligned} &\frac{\partial\, \mathrm{Q}^{\mathrm{s}}_{a_1,\dots,a_M}}{\partial a_m}(x) = \frac{\partial \mathbb{I}^{\mathrm{s}}_{\ge a_m}}{\partial a_m}(x) = \frac{\partial \sigma\left(\frac{x-a_m}{\tau}\right)}{\partial a_m} \\ &= \frac{-1}{\tau} \cdot \sigma\left(\frac{x-a_m}{\tau}\right) \cdot \left(1 - \sigma\left(\frac{x-a_m}{\tau}\right)\right) \\ &= \frac{-1}{\tau} \cdot \mathbb{I}^{\mathrm{s}}_{\ge a_m}(x) \cdot (1 - \mathbb{I}^{\mathrm{s}}_{\ge a_m}(x)) \end{aligned}$$

for all $1 \le m \le M$. This follows from the chain rule and the fact that the derivative of the sigmoid function is given by

$$\frac{d\sigma}{dx}(x) = \sigma(x) \cdot (1 - \sigma(x))$$

for all $x \in \mathbb{R}$. Therefore, the partial derivative of the soft quantization function with respect to a threshold behaves itself as the derivative of a sigmoid function, i.e. it is smooth and non-vanishing but it tends to 0 as x moves away from the threshold.

Similarly, since $(\mathrm{Q}^{\mathrm{bw,s}}_{a_1,\dots,a_M})_m = \mathbb{I}^{\mathrm{s}}_{\ge a_m}$, the partial derivative of the i-th component of the bitwise quantization with respect to a_m vanishes for $i \ne m$, while

$$\frac{\partial (\mathrm{Q}^{\mathrm{bw,s}}_{a_1,\dots,a_M})_m}{\partial a_m}(x) = \frac{-1}{\tau} \cdot \mathbb{I}^{\mathrm{s}}_{\ge a_m}(x) \cdot (1 - \mathbb{I}^{\mathrm{s}}_{\ge a_m}(x)).$$

D. Notation for Quantization Layer

A quantization layer quantizes each input feature independently. This appendix introduces the notation for a quantization layer. Let $K \in \mathbb{N}$ be the number of features and consider for each feature $1 \le k \le K$ a quantization function $Q_k : \mathbb{R} \to \{v^{(k)}_0, \dots, v^{(k)}_{M_i}\}$, where $M_k \in \mathbb{N}$ is the number of

thresholds for feature k and $v_0^{(k)}, \ldots, v_{M_k}^{(k)}$ are the quantized values. Note that $v_m^{(k)} \in \mathbb{R}$ or $v_m^{(k)} \in \{0,1\}^{M_k}$ for all $1 \leq m \leq M_k$. Then the function defined by the quantization layer is given by

$$Q = (Q_1, \ldots, Q_K) : \mathbb{R}^K \to \prod_{k=1}^{K} \{v_1^{(k)}, \ldots, v_{M_k}^{(k)}\} \subset \mathbb{R}^{N'}$$

If all quantization functions are soft quantization functions, the layer is called a soft quantization layer. Note that since soft quantization is differentiable with respect to the thresholds, so is the soft quantization layer. In this case, the number of features, the number of thresholds and the temperature parameters are hyperparameters of the layer, while the thresholds are the parameters that can be learned.

E. Deployment Experiments

For the evaluation of energy and latency overhead we deployed our approach on a microcontroller and measured processing time and energy consumption. We utlized a `Walter` Internet of Things (IoT) system-on-module[5] equipped with an ESP32-S3 CPU. The ESP32-S3 is used in multiple microcontrollers, and the code can easily be transferred to other hardware. We use the Power Profiler Kit II for power profiling. Measurements were taken at a static voltage supply of 3,700 mV, which is commonly used in lithium-ion batteries.

E.1. Encoding

We implemented an exemplary encoder for bit widths of 2, 3, and 4 and deployed it on the microcontroller. As shown in Table 3 the latency experiments show that encoding the input features takes only microseconds and consumes only microjoule, which are negligible compared to the time it takes to measure or transmit the respective data.

Bit Width	Number of Features	Encoding Latency (μs)	Energy Consumption (μAh)	Energy Consumption (μJ)
2	6	10	0.00016	2.11
2	81	50	0.00073	9.79
3	6	10	0.00016	2.11
3	81	70	0.00102	13.54
4	6	10	0.00016	2.14
4	81	70	0.00112	15.62

Table 3: Latency and energy requirements for encoding of input features to bit widths 2, 3, and 4 on a micrcontroller with ESP32-S3 CPU.

Memory: The memory overhead of the encoding functionalities including the thresholds amounts to 433 bytes for a 4-bit compression of 81 features or 203 bytes for a 2-bit compression of 6 features. The numbers are given for the source code compiled with Arduino IDE and written to a ESP32-S3 based MCU. For such a device, again, the overhead is negligible as it has 8 MB of Flash memory. But also for smaller devices, e.g. the Arduino Uno that has only 32 KB of flash, this encoding approach can easily be added to most existing approaches.

E.2. Transmission

We chose to evaluate the data transmission energy and latency requirements in the LTE-M network with the CoAP message protocol, reflecting a widely used setup in the IoT domain. More specifically we used an IoT SIM card from `q.Beyond`, a self-hosted CoAP server running with the `aiocoap`[6] GitHub repository, and required an acknowledgment for all messages. Notably, measuring network

[5] `https://www.quickspot.io`

[6] `https://github.com/chrysn/aiocoap`

latency is a highly complex task and is dependent on many factors. The numbers presented in Table 4 only display an example and are not universal. However, the main purpose is to showcase that the possible energy savings when reducing network latency outweigh the cost of simple encoding.

Byte size	Latency (ms)	Energy Consumption (μAh)	Energy Consumption (μJ)
8 bytes	21	0.6733	8968.34
32 bytes	140	4.0779	54318.13
128 byte	250	8.4043	111945.38

Table 4: Energy and latency requirements for sending different amounts of data via CoAP protocol from a microcontroller equipped with ESP32-S3 CPU.

F. Datasets

All datasets are downloaded using the API from OpenML [44].

Dataset	**Short Name**	**Source**	N	K	$\bar{C}$	**OpenML ID**
California Housing	california	[38]	20640	8	8880	43979
Fried	fried	[39, 40]	40768	10	1001	564
Superconduct	superconduct	[42]	21263	81	7449	43174
CPU Activity Database	cpu_act		8192	21	2192	197
Sulfur	sulfur	[41]	10081	6	8636	23515
Wine Quality	wine	[43]	6497	11	241	287

Table 5: Datasets used where N denotes the size of the dataset, K the number of features and $\bar{C}$ the average number of characteristics (i.e. distinct values) per feature.

G. Temperature Schedule Experiments

We performed experiments to compare different schedules for the temperature parameter τ. The different schedules tested are visualized in Figure 5.

The performance of Bitwise Soft-Quantization for different schedules was tested with 2 different setups: First, on the wine dataset for 2 bit quantization and second, on superconduct for 7 bit quantization. Results are in Table 6 and Table 7. In both setups, all temperature schedules were tested on the respective best hyperparameter setup and use an initial τ value of $\tau_{init} = 1$ and the validation MSE of $k = 4$-fold cross-validation is reported. Note that we selected these two setups to include each one setting where Bitwise Soft Quantization works well, resp. badly. We also tried higher values for τ_{init}, but that led to worse results. In total, the temperature schedule has an influence, but the exponential schedule together with the options $\tau_{end} = 0.0001$ and $\tau_{end} = 0.001$ that we used in the paper works fine.

Schedule	τ_{end} 0.0001	0.001	0.01
Linear	0.6315	0.6515	0.6421
Warmup	0.6425	0.6456	0.6604
Cosine	0.6401	0.6412	0.6320
Cyclical	0.6526	0.6473	0.6503
Exponential	0.6300	**0.6201**	0.6412

Table 6: Average validation MSE of Bitwise Soft-Quantization on wine dataset for 2 bit quantization for different temperature schedules.

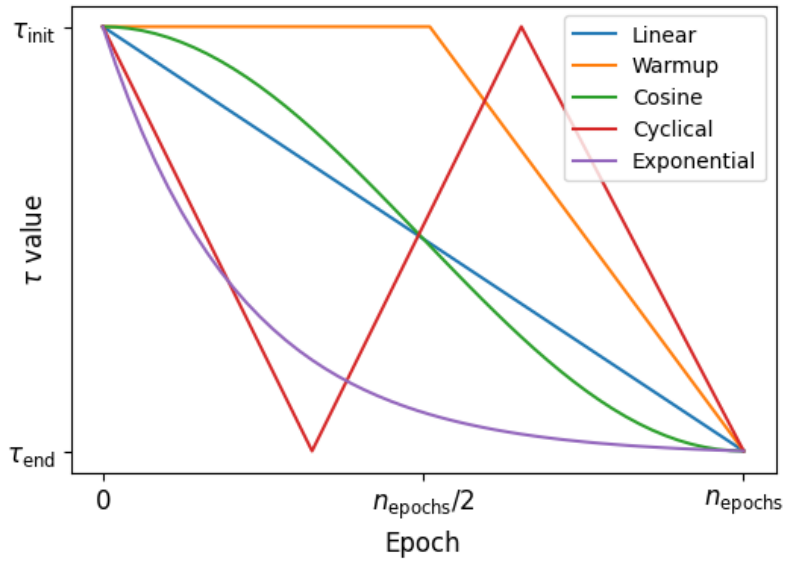


Figure 5: Visualization of different temperature schedules. All schedules start with a temperature value $\tau_{init} = 1$ and end with a temperature value τ_{end} after n_{epochs} epochs.

Schedule	τ_{end} 0.0001	0.001	0.01
Linear	0.1615	0.1790	0.1672
Warmup	0.1848	0.1741	0.1737
Cosine	0.1335	0.1336	0.1449
Cyclical	0.1595	0.1636	0.1896
Exponential	0.1197	**0.1112**	0.1230

Table 7: Average validation MSE of Bitwise Soft-Quantization on superconduct dataset for 7 bit quantization for different temperature schedules.

H. Hyperparameters

The hyperparameter settings used in hyperparameter optimization are listed in Table 8.

Name	Values	Description
dropout rate	[0.0, 0.2, 0.4, 0.5]	dropout rate for neurons during training
learning rate	[0.001, 0.0001]	learning rate for optimizer
hidden layers	[5, 6, 8, 10]	number of hidden layers
hidden neurons	[32, 64, 128, 256, 512, 1024, 2048, 4096, 8192]	number of neurons per hidden layer
num epochs	[30, 50, 70]	number of epochs to train
decrease factor	[0.001, 0.0001]	factor to decrease τ during training

Table 8: Search Space for Hyperparameter Tuning

I. Hardware Setup

All experiments were executed on a server equipped with a NVIDIA GeForce RTX™ 4090 GPU with 24GB GDDR6x VRAM in combination with 8 AMD EPYC™ 9124 CPUs. The code was compiled

with the GCC toolchain version 12.3.0 in combination with PyTorch 2.1.2 for CUDA 12.1. Choices for the hyperparameter were generated with setting a random seed for the `random` package in python.

J. Further Results

The following figures illustrate the distribution of test loss across the individual folds. Note that the scales of the individual datasets also vary across the figures. As the number of bits increases (see Figures 11 and 12), the approaches converge. Furthermore, we add detailed result tables for our experiments and the ablation study.

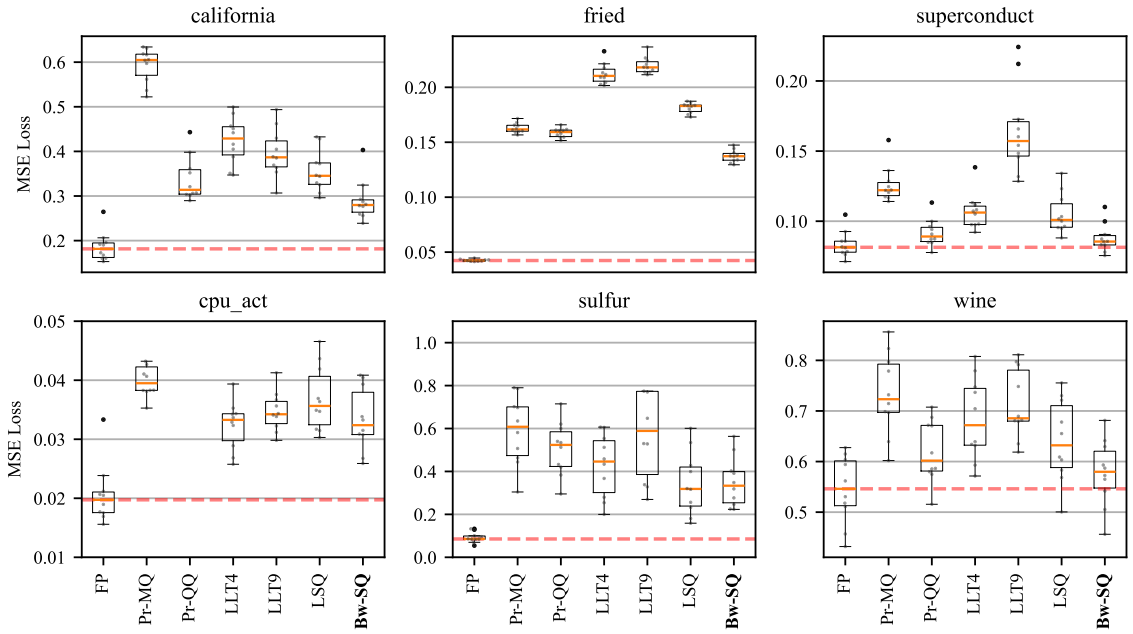


Figure 6: Distribution of the k-fold values for the best average hyperparameter setup per quantization method for a 2-bit quantization level.

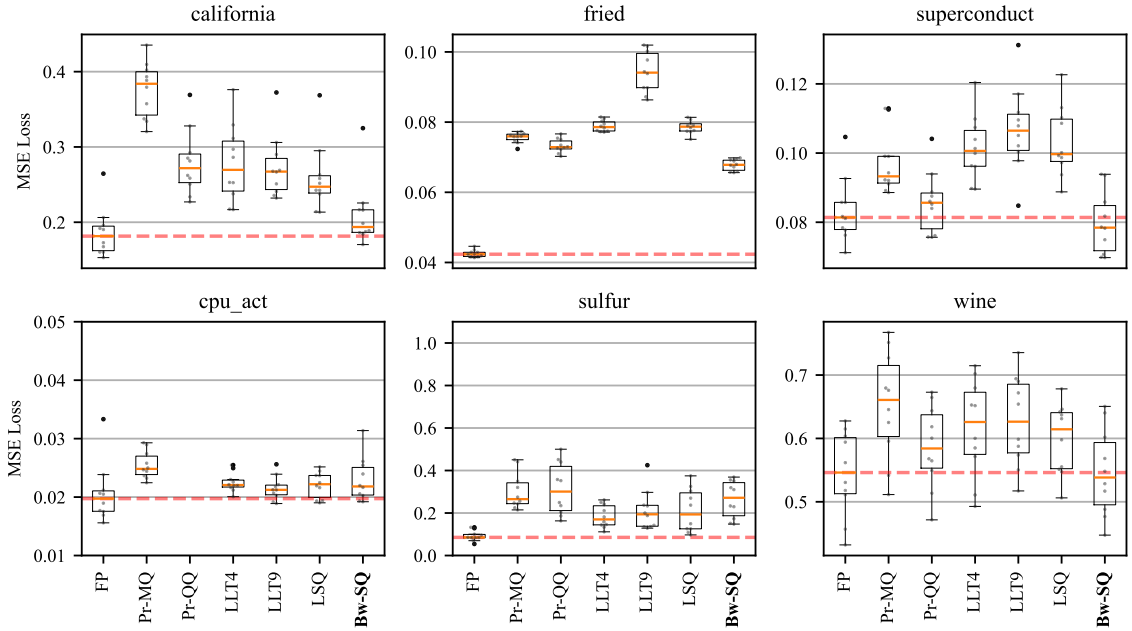


Figure 7: Distribution of the k-fold values for the best average hyperparameter setup per quantization method for a 3-bit quantization level.

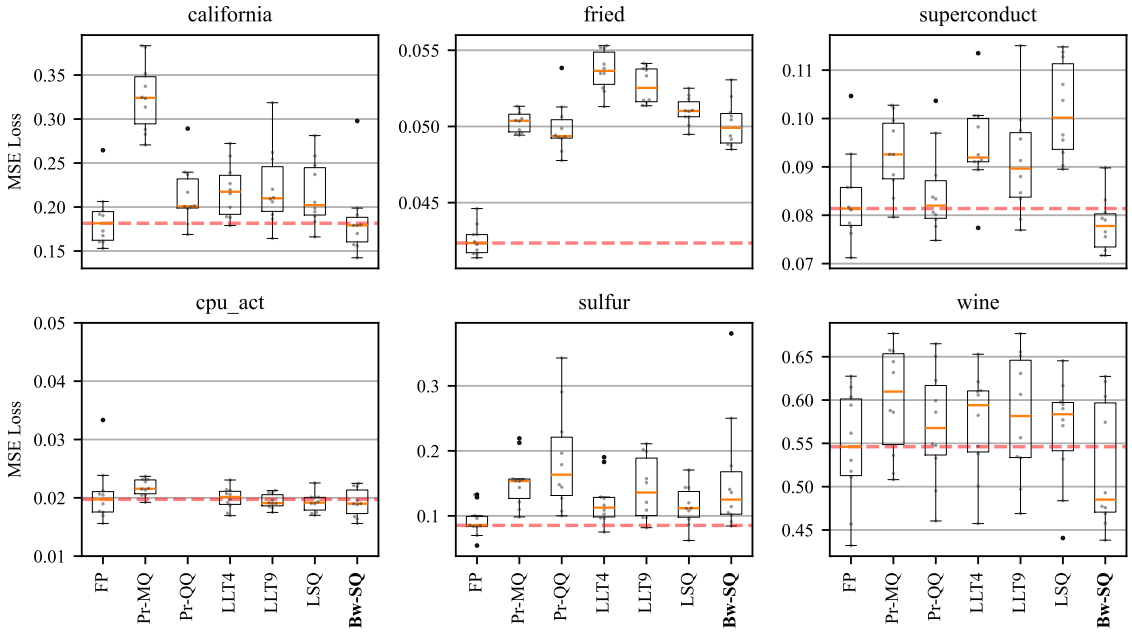


Figure 8: Distribution of the k-fold values for the best average hyperparameter setup per quantization method for a 4-bit quantization level.

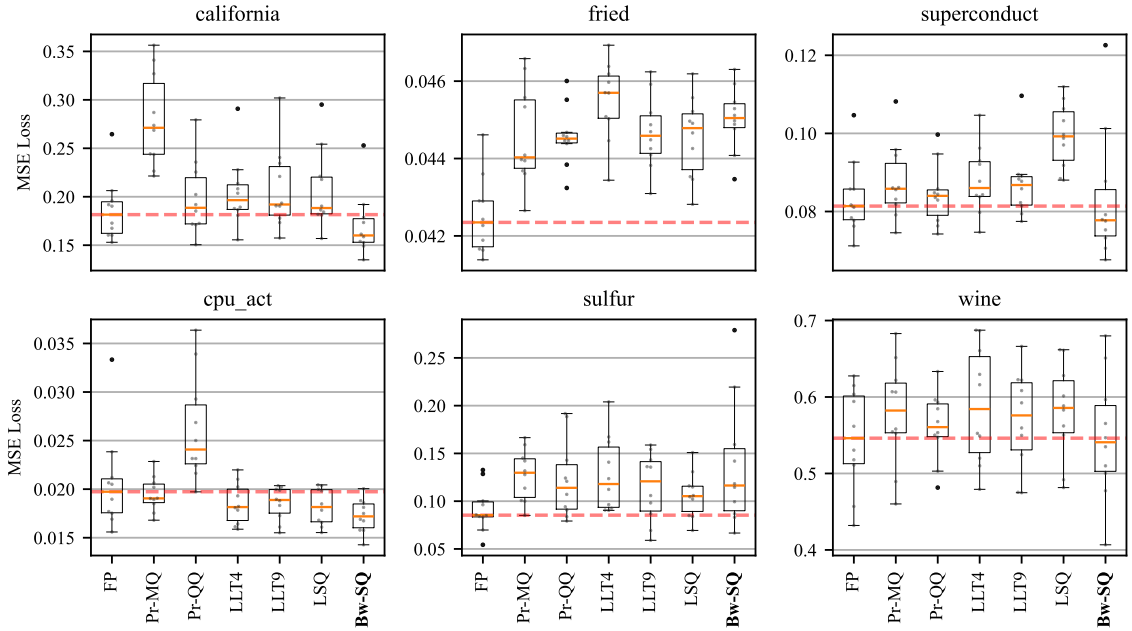


Figure 9: Distribution of the k-fold values for the best average hyperparameter setup per quantization method for a 5-bit quantization level.

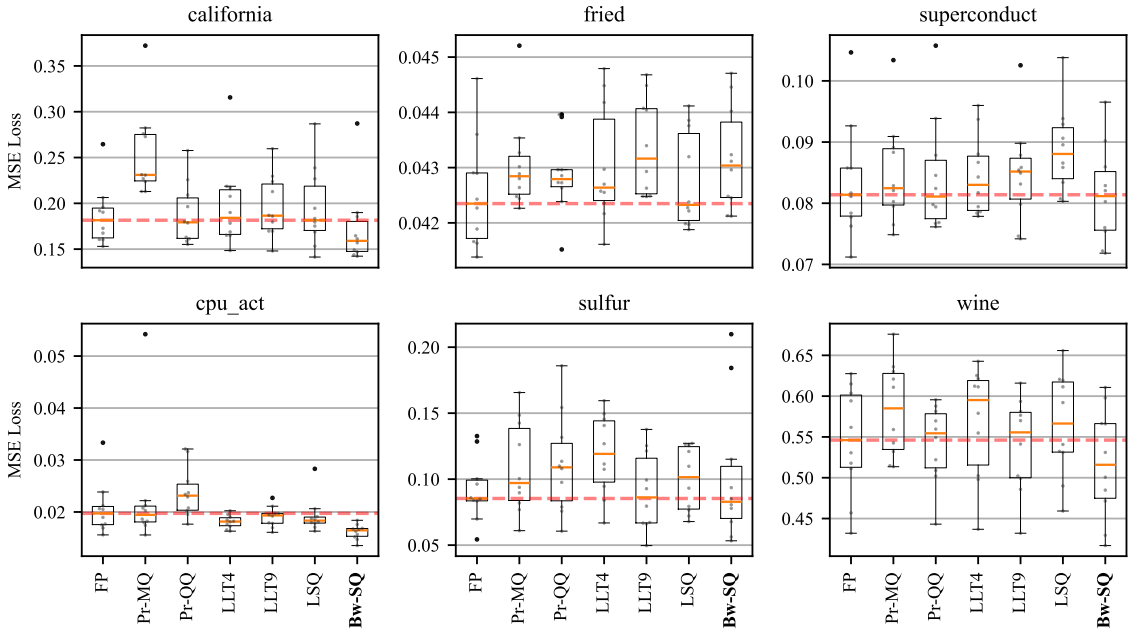


Figure 10: Distribution of the k-fold values for the best average hyperparameter setup per quantization method for a 6-bit quantization level.

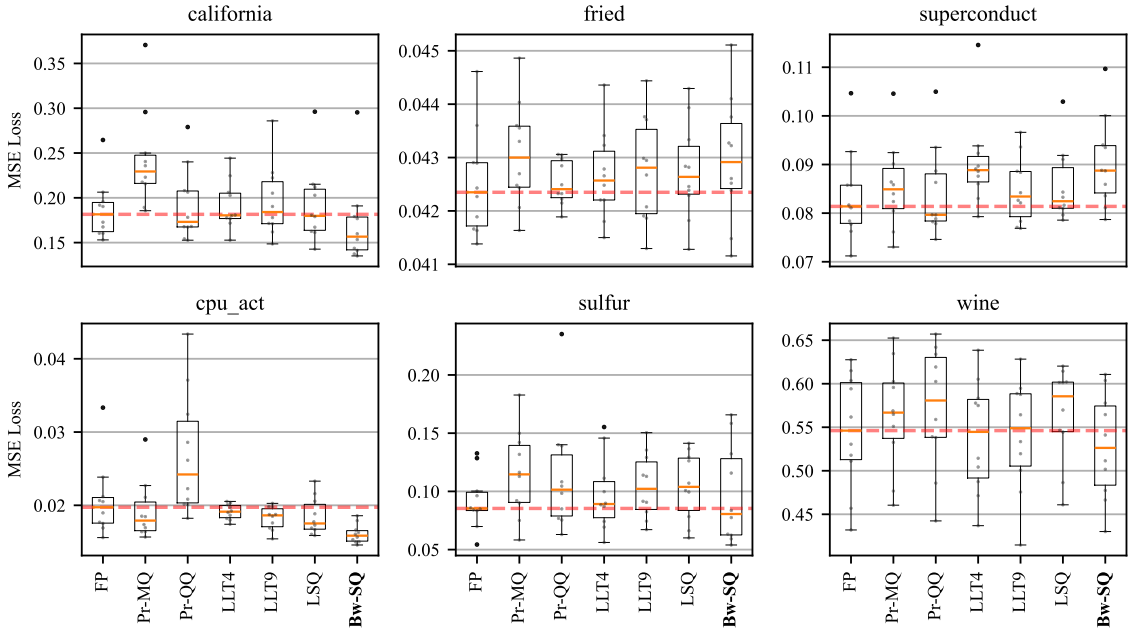


Figure 11: Distribution of the k-fold values for the best average hyperparameter setup per quantization method for a 7-bit quantization level.

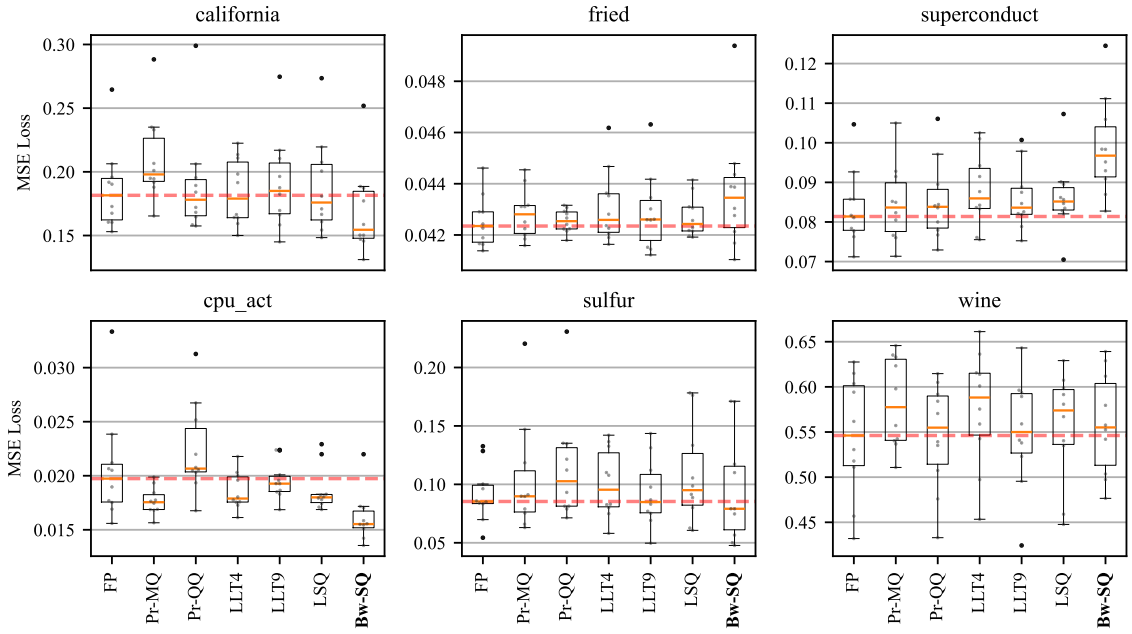


Figure 12: Distribution of the k-fold values for the best average hyperparameter setup per quantization method for a 8-bit quantization level.

dataset	bits	FP	Comparison models					Ours
			Pr-MQ	Pr-QQ	LLT4	LLT9	LSQ	Bw-SQ
	2	0.186	0.593	0.338	0.424	0.396	0.356	**0.290**
	3	0.186	0.376	0.279	0.278	0.273	0.259	**0.210**
	4	0.186	0.326	0.213	0.219	0.222	0.216	**0.185**
california	5	0.186	0.279	0.199	0.204	0.208	0.207	**0.171**
	6	0.186	0.254	0.189	0.197	0.197	0.195	**0.173**
	7	0.186	0.242	0.192	0.192	0.196	0.192	**0.171**
	8	0.186	0.210	0.189	0.184	0.191	0.188	**0.169**
	2	0.043	0.163	0.159	0.212	0.220	0.181	**0.137**
	3	0.043	0.076	0.073	0.079	0.094	0.079	**0.068**
	4	0.043	0.050	**0.050**	0.054	0.053	0.051	0.050
fried	5	0.043	0.045	0.045	0.045	0.045	**0.045**	0.045
	6	0.043	0.043	0.043	0.043	0.043	**0.043**	0.043
	7	0.043	0.043	**0.043**	0.043	0.043	0.043	0.043
	8	0.043	0.043	**0.043**	0.043	0.043	0.043	0.044
	2	0.084	0.126	0.092	0.107	0.164	0.105	**0.088**
	3	0.084	0.097	0.086	0.101	0.107	0.103	**0.080**
	4	0.084	0.093	0.085	0.095	0.091	0.102	**0.078**
superconduct	5	0.084	0.087	0.084	0.088	0.087	0.099	**0.083**
	6	0.084	0.085	0.084	0.084	0.085	0.089	**0.081**
	7	0.084	0.086	**0.084**	0.090	0.085	0.086	0.090
	8	0.084	**0.085**	0.085	0.088	0.086	0.086	0.099
	2	0.021	0.043	0.316	**0.032**	0.035	0.037	0.033
	3	0.021	0.025	0.197	0.022	**0.021**	0.022	0.023
	4	0.021	0.022	0.120	0.020	0.019	0.019	**0.019**
cpu_act	5	0.021	0.019	0.026	0.019	0.019	0.018	**0.017**
	6	0.021	0.023	0.024	0.018	0.019	0.019	**0.016**
	7	0.021	0.019	0.027	0.019	0.018	0.018	**0.016**
	8	0.021	0.018	0.022	0.019	0.020	0.019	**0.016**
	2	0.092	0.591	0.505	0.427	1.490	**0.343**	0.350
	3	0.092	0.302	0.314	**0.184**	0.212	0.212	0.264
	4	0.092	0.152	0.186	0.123	0.143	**0.115**	0.158
sulfur	5	0.092	0.127	0.122	0.128	0.115	**0.106**	0.137
	6	0.092	0.109	0.112	0.119	**0.091**	0.100	0.102
	7	0.092	0.115	0.113	0.098	0.105	0.105	**0.097**
	8	0.092	0.104	0.114	0.101	**0.093**	0.108	0.096
	2	0.545	0.734	0.620	0.684	0.713	0.640	**0.577**
	3	0.545	0.652	0.587	0.616	0.627	0.599	**0.547**
	4	0.545	0.600	0.571	0.573	0.581	0.565	**0.524**
wine	5	0.545	0.578	0.561	0.589	0.570	0.581	**0.547**
	6	0.545	0.584	0.542	0.568	0.540	0.565	**0.518**
	7	0.545	0.564	0.572	0.539	0.541	0.565	**0.528**
	8	0.545	0.582	**0.546**	0.576	0.550	0.556	0.559

Table 9: Average MSE for selected hyperparameter setting per dataset and bit width. Minimal values of baseline and our models per row are bold, values within a range of 2.5% from the minimal value are underlined.

dataset	bits	FP	Comparison models					Ours
			Pr-MQ	Pr-QQ	LLT4	LLT9	LSQ	Bw-SQ
	2	[0.163, 0.210]	[0.565, 0.621]	[0.302, 0.374]	[0.387, 0.462]	[0.357, 0.435]	[0.322, 0.389]	[0.257, 0.323]
	3	[0.163, 0.210]	[0.349, 0.402]	[0.248, 0.310]	[0.241, 0.315]	[0.243, 0.303]	[0.226, 0.291]	[0.178, 0.241]
	4	[0.163, 0.210]	[0.298, 0.354]	[0.188, 0.238]	[0.196, 0.241]	[0.190, 0.254]	[0.189, 0.243]	[0.154, 0.216]
california	5	[0.163, 0.210]	[0.245, 0.313]	[0.171, 0.226]	[0.178, 0.230]	[0.178, 0.239]	[0.177, 0.236]	[0.147, 0.194]
	6	[0.163, 0.210]	[0.219, 0.289]	[0.164, 0.213]	[0.163, 0.232]	[0.172, 0.221]	[0.163, 0.226]	[0.142, 0.204]
	7	[0.163, 0.210]	[0.203, 0.282]	[0.163, 0.222]	[0.172, 0.211]	[0.167, 0.225]	[0.161, 0.223]	[0.137, 0.205]
	8	[0.163, 0.210]	[0.185, 0.234]	[0.160, 0.219]	[0.165, 0.203]	[0.164, 0.217]	[0.161, 0.216]	[0.144, 0.194]
	2	[0.042, 0.043]	[0.160, 0.166]	[0.156, 0.162]	[0.206, 0.219]	[0.215, 0.225]	[0.178, 0.185]	[0.133, 0.141]
	3	[0.042, 0.043]	[0.075, 0.077]	[0.072, 0.075]	[0.078, 0.080]	[0.090, 0.099]	[0.077, 0.080]	[0.067, 0.069]
	4	[0.042, 0.043]	[0.050, 0.051]	[0.049, 0.051]	[0.053, 0.055]	[0.052, 0.054]	[0.050, 0.052]	[0.049, 0.051]
fried	5	[0.042, 0.043]	[0.044, 0.046]	[0.044, 0.045]	[0.045, 0.046]	[0.044, 0.045]	[0.044, 0.045]	[0.044, 0.046]
	6	[0.042, 0.043]	[0.042, 0.044]	[0.042, 0.043]	[0.042, 0.044]	[0.043, 0.044]	[0.042, 0.043]	[0.042, 0.044]
	7	[0.042, 0.043]	[0.042, 0.044]	[0.042, 0.043]	[0.042, 0.043]	[0.042, 0.043]	[0.042, 0.043]	[0.042, 0.044]
	8	[0.042, 0.043]	[0.042, 0.044]	[0.042, 0.043]	[0.042, 0.044]	[0.042, 0.044]	[0.042, 0.043]	[0.042, 0.045]
	2	[0.077, 0.090]	[0.117, 0.135]	[0.085, 0.099]	[0.098, 0.116]	[0.142, 0.187]	[0.095, 0.116]	[0.080, 0.095]
	3	[0.077, 0.090]	[0.091, 0.104]	[0.079, 0.092]	[0.095, 0.108]	[0.098, 0.116]	[0.096, 0.111]	[0.073, 0.086]
	4	[0.077, 0.090]	[0.087, 0.098]	[0.078, 0.091]	[0.088, 0.101]	[0.083, 0.099]	[0.095, 0.109]	[0.074, 0.082]
superconduct	5	[0.077, 0.090]	[0.081, 0.094]	[0.079, 0.090]	[0.082, 0.094]	[0.081, 0.094]	[0.093, 0.105]	[0.071, 0.095]
	6	[0.077, 0.090]	[0.079, 0.091]	[0.078, 0.091]	[0.080, 0.089]	[0.079, 0.091]	[0.084, 0.094]	[0.076, 0.087]
	7	[0.077, 0.090]	[0.079, 0.092]	[0.078, 0.091]	[0.084, 0.097]	[0.080, 0.090]	[0.080, 0.091]	[0.084, 0.097]
	8	[0.077, 0.090]	[0.078, 0.092]	[0.078, 0.092]	[0.081, 0.095]	[0.080, 0.092]	[0.080, 0.093]	[0.090, 0.108]
	2	[0.017, 0.024]	[0.035, 0.051]	[0.283, 0.350]	[0.029, 0.035]	[0.032, 0.037]	[0.033, 0.041]	[0.030, 0.037]
	3	[0.017, 0.024]	[0.024, 0.027]	[0.154, 0.239]	[0.021, 0.024]	[0.020, 0.023]	[0.020, 0.024]	[0.020, 0.026]
	4	[0.017, 0.024]	[0.021, 0.023]	[0.089, 0.151]	[0.019, 0.021]	[0.019, 0.020]	[0.018, 0.020]	[0.017, 0.021]
cpu_act	5	[0.017, 0.024]	[0.018, 0.021]	[0.022, 0.030]	[0.017, 0.020]	[0.017, 0.020]	[0.017, 0.020]	[0.016, 0.018]
	6	[0.017, 0.024]	[0.015, 0.031]	[0.020, 0.027]	[0.017, 0.019]	[0.018, 0.020]	[0.017, 0.022]	[0.015, 0.017]
	7	[0.017, 0.024]	[0.016, 0.022]	[0.021, 0.033]	[0.018, 0.020]	[0.017, 0.019]	[0.017, 0.020]	[0.015, 0.017]
	8	[0.017, 0.024]	[0.017, 0.019]	[0.019, 0.025]	[0.017, 0.020]	[0.018, 0.021]	[0.017, 0.020]	[0.014, 0.018]
	2	[0.075, 0.109]	[0.476, 0.705]	[0.416, 0.595]	[0.322, 0.532]	[-0.478, 3.457]	[0.238, 0.448]	[0.266, 0.433]
	3	[0.075, 0.109]	[0.240, 0.364]	[0.226, 0.401]	[0.145, 0.222]	[0.146, 0.279]	[0.139, 0.285]	[0.202, 0.327]
	4	[0.075, 0.109]	[0.124, 0.181]	[0.129, 0.244]	[0.096, 0.149]	[0.108, 0.177]	[0.093, 0.138]	[0.092, 0.224]
sulfur	5	[0.075, 0.109]	[0.108, 0.146]	[0.093, 0.152]	[0.100, 0.156]	[0.089, 0.140]	[0.089, 0.123]	[0.089, 0.185]
	6	[0.075, 0.109]	[0.084, 0.134]	[0.085, 0.139]	[0.097, 0.141]	[0.069, 0.112]	[0.082, 0.118]	[0.064, 0.141]
	7	[0.075, 0.109]	[0.088, 0.142]	[0.077, 0.148]	[0.075, 0.121]	[0.085, 0.125]	[0.084, 0.125]	[0.067, 0.128]
	8	[0.075, 0.109]	[0.070, 0.138]	[0.080, 0.148]	[0.080, 0.122]	[0.072, 0.113]	[0.077, 0.138]	[0.063, 0.128]
	2	[0.497, 0.593]	[0.676, 0.791]	[0.575, 0.664]	[0.626, 0.743]	[0.664, 0.763]	[0.582, 0.699]	[0.530, 0.625]
	3	[0.497, 0.593]	[0.591, 0.713]	[0.540, 0.634]	[0.561, 0.671]	[0.576, 0.679]	[0.559, 0.639]	[0.498, 0.596]
	4	[0.497, 0.593]	[0.555, 0.645]	[0.524, 0.618]	[0.529, 0.616]	[0.529, 0.633]	[0.520, 0.610]	[0.471, 0.577]
wine	5	[0.497, 0.593]	[0.529, 0.628]	[0.529, 0.593]	[0.534, 0.644]	[0.524, 0.616]	[0.536, 0.625]	[0.489, 0.605]
	6	[0.497, 0.593]	[0.542, 0.625]	[0.508, 0.576]	[0.519, 0.617]	[0.498, 0.581]	[0.520, 0.611]	[0.469, 0.566]
	7	[0.497, 0.593]	[0.519, 0.608]	[0.521, 0.623]	[0.493, 0.586]	[0.494, 0.587]	[0.525, 0.604]	[0.485, 0.572]
	8	[0.497, 0.593]	[0.546, 0.618]	[0.503, 0.588]	[0.530, 0.622]	[0.506, 0.594]	[0.512, 0.599]	[0.519, 0.599]

Table 10: 95%-confidence intervals of 10-fold results for selected hyperparameter settings per dataset and bit width.

dataset	bits	Comparison models			Ours
		SQ	Bw-MQ	Bw-QQ	Bw-SQ
	2	0.288	0.595	0.329	**0.290**
	3	0.217	0.377	0.265	**0.210**
	4	0.214	0.312	0.207	**0.185**
california	5	0.208	0.262	0.182	**0.171**
	6	0.200	0.230	0.177	**0.173**
	7	0.202	0.199	**0.171**	0.171
	8	0.205	0.180	0.173	**0.169**
	2	0.137	0.164	0.158	**0.137**
	3	0.070	0.074	0.073	**0.068**
	4	0.052	0.050	0.050	**0.050**
fried	5	0.048	0.045	0.045	**0.045**
	6	0.046	0.044	**0.043**	0.043
	7	0.047	0.044	0.045	**0.043**
	8	0.047	**0.043**	0.045	0.044
	2	0.094	0.134	0.090	**0.088**
	3	0.088	0.101	0.083	**0.080**
	4	0.087	0.092	0.079	**0.078**
superconduct	5	0.097	0.093	0.086	**0.083**
	6	0.094	0.089	0.082	**0.081**
	7	0.089	0.097	**0.084**	0.090
	8	0.085	0.092	**0.090**	0.099
	2	0.034	0.035	0.160	**0.033**
	3	0.027	**0.022**	0.037	0.023
	4	0.025	0.020	0.025	**0.019**
cpu_act	5	0.023	0.020	0.023	**0.017**
	6	0.025	0.017	0.019	**0.016**
	7	0.024	0.017	0.018	**0.016**
	8	0.025	0.018	0.019	**0.016**
	2	0.348	0.593	0.498	**0.350**
	3	0.305	0.287	0.374	**0.264**
	4	0.175	0.184	0.210	**0.158**
sulfur	5	0.152	**0.120**	0.163	0.137
	6	0.129	**0.099**	0.126	0.102
	7	0.134	**0.087**	0.123	0.097
	8	0.123	**0.090**	0.114	0.096
	2	0.610	0.733	0.596	**0.577**
	3	0.584	0.645	0.560	**0.547**
	4	0.567	0.582	**0.514**	0.524
wine	5	0.583	0.519	**0.508**	0.547
	6	0.585	**0.496**	0.509	0.518
	7	0.566	**0.492**	0.500	0.528
	8	0.584	**0.517**	0.561	0.559

Table 11: Ablation Study: Average MSE for selected hyperparameter setting per dataset and bit width. Minimal values are bold, values within a range of 2.5% from the minimal value are underlined.