Quantization Index Modulation-based Reversible Data Hiding in Compressed Neural Network

Jun Hirano, Jonethe Tan Yang, Fathin Acyuta Makarim, Daham Jayasinghe, KokSheik Wong†
School of Information Technology, Monash University Malaysia, Malaysia.
Emails: {jhir0003, jtan0396, fmak0006, jjay0020}@student.monash.edu, wong.koksheik@monash.edu

***Abstract*—As deep neural networks are increasingly deployed in diverse applications, ensuring the integrity of these models during storage and transmission has become critical. In such scenarios, model compression is not only essential for reducing size and bandwidth but also serves as a practical stage to integrate protection mechanisms. Such protection mechanisms can be realized using data hiding technology. In this work, we extend neural network coding NNCodec (ISO/IEC 15938 17) by introducing a reversible data hiding method. We insert a message using histogram-based quantization index modification in the quantized weights, then package the modified weights together with the syntax elements and headers into a fully NNCodec compliant bitstream. To the best of our knowledge, we are the first to propose a reversible data hiding method specifically for NNCodec. Experiments were carried out on three widely used architectures, namely ResNet18, VGG16, and MobileNetV2, to verify the basic performance of the proposed reversible data hiding method for NNCodec coded bitstream. The source code is available on GitHub for reproducibility.[1]**

***Index Terms*—NNCodec; QIM; Compression; Neural Network Compression.**

I. Introduction

Deep learning models are increasingly being deployed in a wide spectrum of applications, including critical fields such as healthcare [1], finance [2], and autonomous systems [3]. As these models become more valuable and impactful to our well-being, protecting their integrity becomes a crucial task to ensure that the received models operate as they were originally trained or programmed to do. To this end, various model watermarking and protection methods have been proposed. For example, Minoru et al. [4] propose a quantization-based watermarking method using weights sampled from the fully connected layers. On the other hand, Fan et al. [5] address watermark ambiguity attacks (i.e., an attempt to cast doubt on ownership) and network modification by introducing a digital passport, where the to-be-protected deep neural network (DNN) model is trained so that its inference performance of an original task is significantly deteriorated when using forged passports.

On the other hand, as neural networks grow in complexity, efficient storage and transmission become increasingly important. One approach is to perform *model pruning*, where redundant or unimportant parameters are removed to reduce model size [6], [7]. As an alternative, and from a different perspective, the traditional multimedia compression framework—comprising transformation, quantization, and entropy coding—can be adapted to compress neural networks effectively. When employed together, pruning and compression can significantly reduce model size, though often at the cost of a slight degradation in performance. In this work, we focus on the newly finalized international standard for neural network model compression, namely the **Neural Network Coding (NNCodec)** standard, formally known as *ISO/IEC 15938-17:2024* [8]. We put forward a method to hide data directly into a NNCodec encoded bitstream.

Our proposed data hiding method can be utilized to protect the integrity and the authenticity of compressed neural network models. This is achieved by embedding a form of certification or watermark into the model weights during compression, so that model owners can ensure that the deployed or distributed model is original and unaltered. In addition, since our proposed method is based on quantized indexed modulation (QIM) and histogram shifting, it can achieve *reversibility*. Specifically, our method can be deployed to implement a "try-before-you-buy" business model. Specifically, a trained neural network model can be degraded by drastically changing selected weight parameters, where their original values are hidden in the network using our method. This degraded model is then distributed freely so that potential buyers can try it out. When a purchase transaction is completed, the original weight can be extracted and updated in the model to unleash the full performance of the model.

Our work makes the following contribution: (a) We propose the first data hiding method for neural network model compressed using the ISO/IEC 15938-17 Neural Network Coding (NNCodec) standard; (b) The modified NNCodec bitstream produced by our data insertion method is format compliant, and hence it can be decoded by the NNCodec-compliant decoder without errors; (c) Our histogram-based QIM embedding is applied after URQ quantization, where no retraining or modification of the original network architecture is required, and; (d) Reversible, where the original quantized weight values can be recovered.

[1]https://github.com/Jun0003/NeuralNetworkCompression-HQIM-DataHiding

†This research is supported, in part, by the Fundamental Research Grant Scheme: *Complete Quality Preservation and Reversible Data Hiding - from formulation to application* (Grant No. FRGS/1/2023/ICT07/MUSM/02/1).

II. PRELIMINARY

This section provides a brief overview of the NNCodec, reviewing the elements and components exploited in our method to hide data using the quantized weight parameter values.

A. NNCodec Compression Framework

In its simplest form, a neural network model consists of layers of interconnected units (neurons), where each connection is associated with a weight, i.e., a real-valued parameter that determines the strength of the signal. In essence, a neural network can be viewed as a collection of such weights, which collectively encode the learned knowledge of the model [9].

To facilitate efficient storage and transmission, these weights can be quantized and entropy-encoded, which is the core approach adopted in the NNCodec framework [8]. Specifically, NNCodec takes a trained neural network model (e.g., in TensorFlow format) as input and produces a compressed bitstream as output. The overall processing flow and key components of NNCodec are shown in Fig. 1, with three main components: pre-processing, quantization, and entropy coding.

First, at the pre-processing stage, parameter reduction takes place, where processes such as sparsification, pruning, and/or low-rank decomposition are performed. Next, in the quantization stage, either a uniform reconstruction quantization (URQ) or Trellis-coded quantization (TCQ) takes place to reduce the precision of the parameter values. For the case of URQ, each learned weight w is mapped to a quantized value $\hat{w}$ on a uniform grid with quantization step size Δ:

$$\hat{w} = Q_\Delta(w) = \Delta \cdot \left\lfloor \frac{w}{\Delta} + \frac{1}{2} \right\rfloor,$$

where $\hat{w} \in \{\ldots, -2\Delta, -\Delta, 0, \Delta, 2\Delta, \ldots\}$. In essence, this nearest-neighbor rule assigns each real-valued weight to its closest quantization bin, yielding a maximum absolute quantization error of $\Delta/2$. Interested reader may refer to [10] for more information about TCQ. It is noteworthy that the parameter reduction step is optional. Subsequently, the resulting quantized values are encoded using the adapted version of the context-based adaptive binary arithmetic coding (CABAC) scheme. Together with the syntax elements and headers, the codewords collectively form the output (bitstream) of NNCodec.

NNC achieves a significant compression rate, i.e., reducing model size to under 5% of the original in some cases while maintaining accuracy.

B. Quantized Index Modulation [11]

Quantized index modulation is a straightforward data hiding method, where the final output value depends on the data (bit) to be inserted. Given a value x and the quantizer (divisor) d, the output value is

$$\hat{x} = \begin{cases} sign(x) \times (|x| + 1) & \text{if } m \neq \mod(|x|, 2); \\ x & \text{otherwise.} \end{cases} \quad (1)$$

Here, m is the message bit to be encoded using x, $sign(x)$ is the polarity of x, where mod refers to the modulo function. In more advanced (non-uniform) methods, the divisor varies depending on the range of the input value (e.g., guided by the cumulative frequency density [12]).

C. Histogram Shifting

Histogram shifting is one of the earliest reversible data hiding methods, introduced by Ni et al. [13]. In this technique, the histogram of a set of values (in this work, the quantized weights) is first generated, and the most frequently occurring value w^p is selected to hide data. Without loss of generality, the histogram bins to the right of w^p are shifted one step to the right, i.e., $w \leftarrow w + 1$ for all $w > w^p$, while the values $w < w^p$ remain unchanged. This shifting process creates an empty bin at $w^p + 1$, enabling a simple yet effective reversible data hiding mechanism. In particular, data can then be hidden using a mapping rule, e.g., w^p to represent $m = 1$, and $w^p + 1$ to represent $m = 0$.

For an overview of further developments in reversible data hiding techniques include histogram shifting, see the review article by Wang et al. [14].

III. METHODOLOGY

We modify the original NNCodec's quantization component (i.e., the middle block in Fig. 1 labelled by "URQ") by extending it using a bespoke histogram-based quantization index modulation (H-QIM) method for data insertion. Specifically, when the model weights are quantized into low-bitwidth integers (indices), the histogram of the quantized weights is plotted to identify the most frequent weight w_p, i.e., the histogram's peak (see Fig. 2(a), where $w^p = 4$). Subsequently, all weights $w > w^p$ are updated to $w \leftarrow w^p + 1$ to create an empty bin at $w^p + 1$ (see Fig. 2(b) where $w^p = 4$ and $w^p + 1 = 5$). Our H-QIM method modifies the weight parameters of value w^p based on the message to be encoded. In particular, $w^p(i) \leftarrow w^p(i) + m(i)$, where $w^p(i)$ denotes the i-th weight parameter of value w^p, and $m(i)$ refers to the i-th bit of the message bitstream m to be injected. The process is repeated for each i until either all weight parameters of value w^p are exhausted, or when the entire message bitstream m is injected. The modified quantized weight values (now containing message m, see Fig. 2(c)) are sent to the DeepCABAC component for further processing to output a NNCodec compliant bitstream.

To extract the inserted message m, the opposite flow of processes in the encoding pipeline is taken. Specifically, the decoder is modified to take two additional input arguments, i.e., w^p and $w^p + 1$. After DeepCABAC decoding, the collection of quantized-modified weights is obtained. Using the same bit association rule, each occurrence of w^p or $w^p + 1$ in the matrix is mapped back to a bit (i.e., '1' and '0', respectively) and concatenated to reconstruct the message bitstream m. After message extraction, the original histogram can be restored by updating $w \leftarrow w - 1$ for all $w > w^p$, hence, achieving the reversible property.

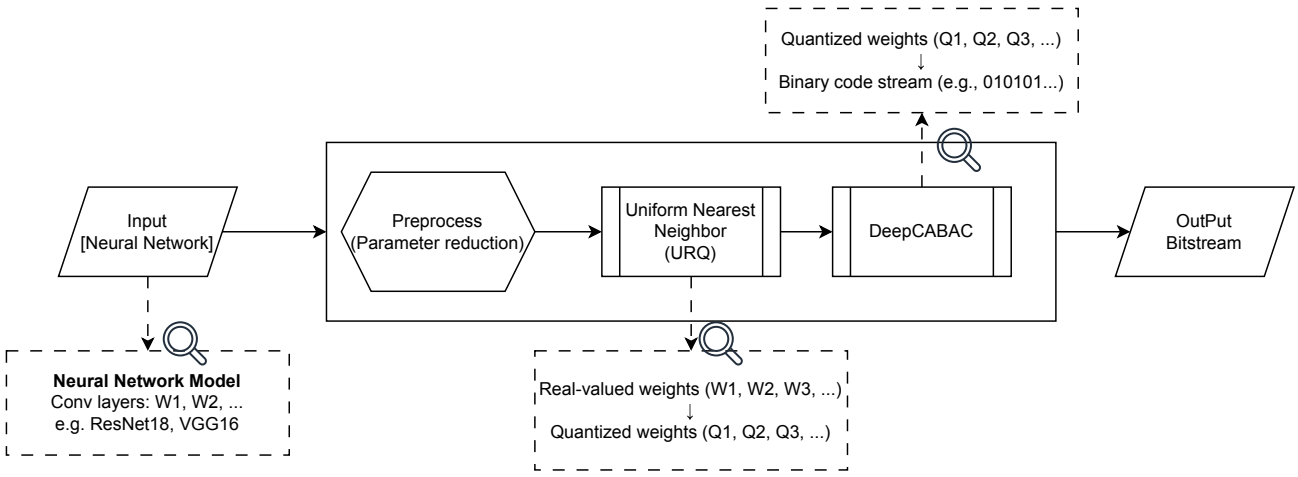


Fig. 1. Pipeline of the encoding process for deep neural network.

IV. Experiments and Results

As a proof of concept, the source codes provided by Fraunhofer [8] are modified to implement the proposed data hiding method. We considered three distinct neural network architectures, namely, VGG16 [15], MobileNetV2 [16], and ResNet18 [17]. Recall that VGG16 is a deep convolutional neural network characterized by its simplicity, using a stack of convolution layers 3×3 followed by fully connected layers. On the other hand, MobileNetV2 is a lightweight model designed for mobile and embedded applications, using separable convolutions in depth and inverted residual blocks to reduce computational complexity. Last but not least, ResNet18 utilizes residual learning with shortcut connections, which allows deeper networks to be trained by addressing the problem of vanishing gradient. These models are trained and evaluated on the CIFAR-10 datasets [18] for object recognition. All models were implemented using the PyTorch deep learning framework and trained within a Google Colab environment. In this work, the classification accuracy is chosen to be assessed using standard top-1 accuracy metrics on a held-out test set. Here, we use a randomly generated bitstream as the message m to be inserted. Note that here, for experimental purposes, the *parameter reduction* component within the pre-processing stage of NNCodec is not activated. This decision is made so that the results can objectively assess the impact made solely by the proposed H-QIM data hiding method. The experiment for each scenario is repeated five (5) times, and the average results are reported.

A. Payload

First, we investigate the maximum number of bits that can be injected into each model. We found that there are $134,301,514$ $2,270,794$, and $11,191,242$ parameters in VGG16, MobileNet and ResNet18, respectively, which will undergo the URQ quantization process within the compression pipeline stipulated by NNCodec. Among these parameter values, the peak parameter value occurs 25,932,685, 13,807, and 190,387 times for VGG16, MobileNet, and ResNet18, respectively, and these numbers are referred to as the payload for each model. For the rest of our discussions, we use payload as the reference and consider a randomly generated message of length 10%, 50%, and 100% of the payload. It is verified that the inserted data can be extracted from the modified quantized parameter values without any errors.

B. File Size

The model size is recorded in Table I for three stages: before compression (Raw), after compression using NNCodec (Comp), and after data hiding post-compression (DaCo). Here, DaCo(x) refers to the scenario where a random message of length $x\%$ of the payload is hidden.

As expected, model size is significantly reduced after compression, confirming the effectiveness of NNCodec. Interestingly, the file size after data insertion is smaller than its original-compressed counterpart (Comp), except for the case of DaCo(100). A potential explanation for this phenomenon is that the minor modifications introduced during data insertion may result in a distribution (of the weights) that is slightly more favorable for entropy coding (i.e., less entropy). However, as the insertion rate increases (from 10% to 50%, and then to 100%), the file size increases steadily. This increasing trend is attributed to the higher level of randomness introduced into the weight distribution, which reduces the compression efficiency of DeepCABAC. These outcomes also suggest that the original quantized parameters are of higher entropy in comparison to their modified counterparts (due to data insertion), which leads to higher accuracy (performance) by the original compressed model.

C. Model Performance

In this section, we investigate the impact of compression and data insertion on model performance. Table II records the accuracy of all three models under different scenarios. It is observed that, when compression is carried out, the performance of each model (i.e., accuracy) drops, which is an

TABLE I
FILE SIZE [KBYTES] REQUIRED TO STORE THE NETWORKS AT DIFFERENT STAGES: RAW (BEFORE COMPRESSION), COMP (AFTER COMPRESSION), AND DACO (DATA INSERTED AFTER COMPRESSION) WITH DIFFERENT INSERTION RATES.

Model	Raw	Comp	DaCo(10)	DaCo(50)	DaCo(100)
VGG16	537,218	56,210	53,066	54,481	56,211
MobileNetV2	9,195	2,512	2,511	2,512	2,512
ResNet18	44,806	9,395	9,376	9,384	9,395

TABLE II
STATISTICS OF MODEL ACCURACY FOR VARIOUS SCENARIO OF COMPRESSION AND DATA INSERTION PROCESSES.

Model	Payload	Type	Mean	Median	Min/Max	Variance (10^{-3})
VGG16	N/A	Raw	88.660	88.660	88.660 / 88.660	0.000
	N/A	Comp	88.840	88.840	88.840 / 88.840	0.000
	0%	Bins Shifted	81.940	81.940	81.940 / 81.940	0.000
	10%	Injected	80.644	80.640	80.610 / 80.690	0.880
	50%	Injected	80.182	80.210	80.130 / 80.220	2.270
	100%	Injected	79.224	79.220	79.130 / 79.380	10.230
	N/A	Reconstructed	88.840	88.840	88.840 / 88.840	0.000
MobileNetV2	N/A	Raw	92.380	92.380	92.380 / 92.380	0.000
	N/A	Comp	92.360	92.360	92.360 / 92.360	0.000
	0%	Bins Shifted	91.270	91.270	91.270 / 91.270	0.000
	10%	Injected	91.272	91.270	91.260 / 91.280	0.070
	50%	Injected	91.270	91.270	91.260 / 91.280	0.010
	100%	Injected	91.274	91.270	91.270 / 91.280	0.030
	N/A	Reconstructed	92.360	92.360	92.360 / 92.360	0.000
ResNet18	N/A	Raw	87.520	87.520	87.520 / 87.520	0.000
	N/A	Comp	87.510	87.510	87.510 / 87.510	0.000
	0%	Bins Shifted	74.350	74.350	74.350 / 74.350	0.000
	10%	Injected	74.172	74.190	74.120 / 74.230	2.120
	50%	Injected	74.028	74.010	73.970 / 74.090	2.220
	100%	Injected	74.040	74.070	73.920 / 74.110	5.900
	N/A	Reconstructed	87.510	87.510	87.510 / 87.510	0.000

expected outcome since the weights are of lower precision due to the quantization process in NNCodec [19]. Results suggest that the impact of lossy compression on model performance is minimal for MobileNetV2 (-0.02%) ResNet18 (-0.01%), and VGG16 ($+0.18\%$).

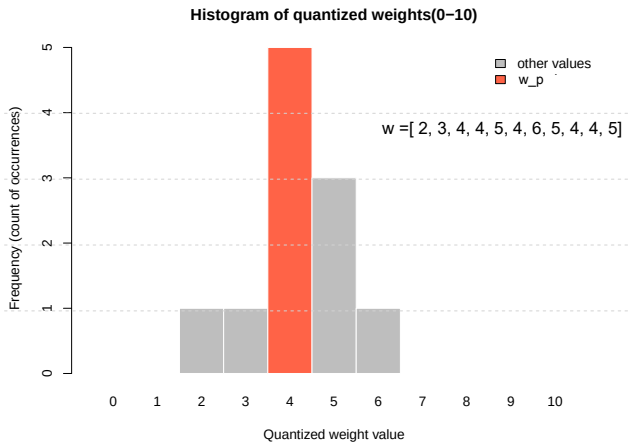
The performance drops further when data is injected, where the parameter values drift further away from the corresponding original quantized values. Specifically, at the insertion rate of 10% payload, the drop in accuracy for MobileNetV2 is minimal (-1.11%), suggesting that data insertion in the already-degraded model (due to compression) has little impact on its performance. However, a more significant drop in accuracy is observed for both VGG16 (-8.01%) and ResNet18 (-13.34%) across all embedding rates. This disparity arises from the models' differing architectural sensitivities to weight perturbations. MobileNetV2's depthwise-separable convolutions combined with comprehensive batch-normalization effectively suppress minor distortions, whereas VGG16 and ResNet18, built from long sequences of standard convolutional layers with substantially higher parameter counts, tend to amplify even small perturbations as they pass through successive layers, resulting in more pronounced accuracy degradation.

Interestingly, for all three models, when more data is inserted into the model, the additional data (e.g., from 10% to 50% payload) does not impact the performance much. In other words, the most significant impact is observed in the case of 10% insertion rate, while the performance loss experienced in the case of 50% and 100% insertion rates is about the same as that of the 10% insertion rate. These suggest that the primary source of accuracy loss is not caused by the inserted message, but rather the bin shifting operation that alters the distribution of the quantized weights. To confirm this conjecture, we identified the peak value w^p and right-shifted the bins of values $w > w^p$ without inserting any messages. Results (rows with 0% payload in Table II) confirm our conjecture, where a significant drop in accuracy is caused by shifting the bins (e.g,. for ResNet18, the accuracy drops from 87.510 for "Comp" to 74.350 for "Bins Shifted").

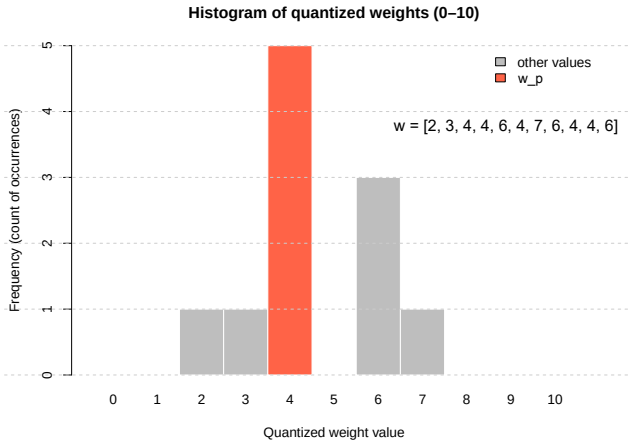
It is noteworthy that, although our proposed data insertion method is reversible, the reports mean results for "Reconstructed" do not match those of "Comp". It is because the models are trained and tested five (5) times, and the average results are reported. Nonetheless, the results are very similar.

V. LIMITATIONS

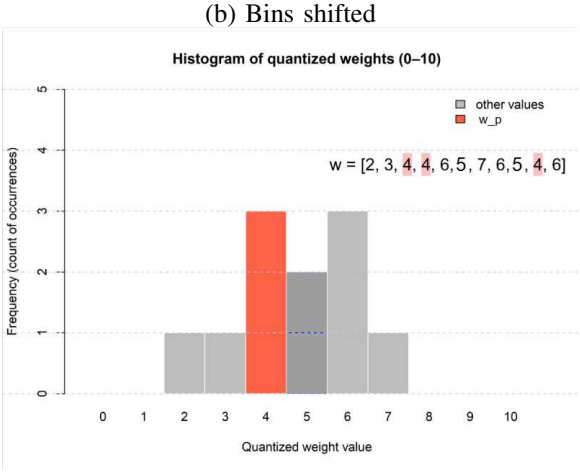
While the basic operations of a reversible data insertion method are achieved, our work has a few limitations. First, the current H-QIM approach lies in its dependency on the number of available w^p values within the model's quantized weight distribution. Since data is inserted using the peak parameter value, the payload is inherently constrained by the frequency and distribution of w^p. As a result, certain models may only allow a short message to be inserted, which limits



(a) Original distribution



(b) Bins shifted



(c) Final distribution

Fig. 2. Illustration of the histogram of weights at different stages of the data insertion process. (a) shows the original distribution, where (b) shows the histogram after the bins of value $> w^p$ being shifted to the right by unity, and (c) shows the final distribution after data insertion.

the practicality of the method for high-capacity data hiding scenarios. To address this, a potential direction for future work is to extend the H-QIM method to support multiple w^p values simultaneously. In addition, the current work only explored VGG16, MobileNetV2, and ResNet18, and the impact of data insertion using H-QIM into NNCodec compressed models on performance is not investigated.

Second, the reversibility property comes at the cost of accuracy degradation during data injection. As reported in Section IV-C, the accuracy drop is primarily caused by the bin shifting mechanism itself, rather than the payload. This means that even a small amount of embedded data may incur a similar level of accuracy loss as larger payloads, which can be inefficient for scenarios requiring only minimal data insertion.

VI. Conclusions

In this work, the Neural Network Encoder/Decoder compression framework is enhanced by implementing a data insertion mechanism. Specifically, the quantized parameter weight values produced by the URQ component in the NNCodec compression pipeline are modified using histogram shifting to insert data. Our method is the first of its kind to insert data into NNCodec-compliant bitstreams. It is also reversible, where the original weights can be reconstructed. Experiments were carried out using VGG16, MobileNetV2, and ResNet18 for object recognition from the CIFAR-10 dataset.

As for future works, we plan to explore alternative data hiding mechanisms that utilize neural network weights beyond histogram shifting and Quantization Index Modulation approaches. This includes the investigation of methods that can offer higher embedding capacity while maintaining negligible accuracy degradation. Performance for a broader range of neural network architectures will also be investigated, including EfficientNet and Vision Transformers

References

[1] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017.

[2] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, no. 2, pp. 654–669, 2018.

[3] K. Muhammad, A. Ullah, J. Lloret, J. Del Ser, and V. H. C. de Albuquerque, "Deep learning for safe autonomous driving: Current challenges and future directions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4316–4336, 2021.

[4] M. Kuribayashi, T. Tanaka, S. Suzuki, T. Yasui, and N. Funabiki, "White-box watermarking scheme for fully-connected layers in fine-tuning model," in *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*, IHMMSec '21, (New York, NY, USA), p. 165–170, Association for Computing Machinery, 2021.

[5] L. Fan, K. W. Ng, C. S. Chan, and Q. Yang, "Deepipr: Deep neural network ownership verification with passports," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 6122–6139, 2022.

[6] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1135–1143, 2015.

[7] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[8] D. Becking, P. Haase, H. Kirchhoffer, K. Müller, and W. Samek, "NNCodec: An open source software implementation of the neural network coding ISO/IEC standard," in *ICML 2023 Workshop Neural Compression: From Information Theory to Applications*, 2023.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[10] P. Haase, H. Schwarz, H. Kirchhoffer, S. Wiedemann, T. Marinc, A. Marban, K. Müller, W. Samek, D. Marpe, and T. Wiegand, "Dependent scalar quantization for neural network compression," in *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 36–40, 2020.

[11] B. Chen and G. Wornell, "Quantization index modulation: a class of provably good methods for digital watermarking and information embedding," *IEEE Transactions on Information Theory*, vol. 47, no. 4, pp. 1423–1443, 2001.

[12] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: efficient finetuning of quantized llms," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, (Red Hook, NY, USA), Curran Associates Inc., 2023.

[13] Z. Ni, Y. Shi, N. Ansari, and W. Su, "Reversible data hiding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, pp. 354–362, Mar. 2006.

[14] J. Wang and B. Ou, "Video reversible data hiding: A systematic review," *Journal of Visual Communication and Image Representation*, vol. 98, p. 104029, 2024.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 2009.

[19] ISO/IEC JTC 1/SC 29, "Information technology — multimedia content description interface — part 17: Neural network coding." https://www.iso.org/standard/80387.html, 2022. ISO/IEC 15938-17:2022.