Low-Precision Neural Network Decoding of Polar Codes

Igor Wodiany, Antoniu Pop
School of Computer Science, The University of Manchester, Manchester, United Kingdom
Email: igor.wodiany@student.manchester.ac.uk, antoniu.pop@manchester.ac.uk

***Abstract*—Neural Network (NN) polar decoders have been getting much attention as a viable replacement for conventional decoders in 5G New Radio (NR). Despite scalability issues, the NN-based decoder is a promising technology as it can improve the latency of the standard Successive Cancellation (SC) decoder. It was shown that the Neural Successive Cancellation (NSC) decoder has an improved theoretical latency compared to the standard SC decoder. However, in contrast to SC, the NSC decoder uses large floating-point weight matrices which do not fit in CPU caches, leading to higher energy usage and lower computational performance due to the increased memory traffic. Additionally such higher memory requirement would be expensive to implement in hardware and require complex floating-point arithmetic. This paper presents a new low-precision NN decoder that can replace memory-heavy NN decoders inside the NSC decoder. We show that up to 54 times weights' size reduction can be achieved with the wireless performance degradation varying between 0.1dB and 0.4dB compared to the floating-point implementation. Moreover, we show a reduction of up to respectively 438$\times$ and 555$\times$ in L1 and L2 data cache misses in our prototype software implementation.**

***Index Terms*—Polar codes, deep learning, weights quantization, neural network.**

I. Introduction

Polar codes, proposed by Arikan [1], have gained much attention, because of the mathematically proven channel achieving performance and a low complexity of code construction and decoding algorithms. Polar codes were adopted by 3GPP as a channel coding technique for control channels in 5G New Radio (NR) [2]. Unfortunately the simple, low complexity successive cancellation (SC) decoder suffers from being inherently serial and does not achieve a sufficient wireless performance for small codeword sizes, where the more complex successive cancellation list (SCL) decoder [3] is required. Much work has been done to improve the latency and throughput of these decoders [4], [5], [6], however their serial nature still remains the problem.

Recently a novel deep learning (DL) decoder based on an artificial neural network (NN) [7] has shown a competitive wireless performance while also enabling a highly parallel one-shot decoding. Unfortunately this solution does not scale well for large message sizes as the training time becomes prohibitive. To alleviate this problem a partitioned neural network decoder was proposed [8]. It uses a *Belief Propagation (BP)* algorithm to connect a partitioned NN decoder (NND) allowing decoding of large codewords in exchange for an increased complexity. To reduce the penalty of partitioning the decoder, the neural successive cancellation (NSC) decoder was proposed [9]. It replaces the BP algorithm of the partitioned decoder with the SC decoder that results in decreased latency. The analogy can be drawn to the simplified successive cancellation (SSC) decoder [10] where final stages of the decoder are replaced with NN decoders of size 16.

Despite its clear benefits the proposed NSC decoder uses floating-point values for both training and inference. As much as 700 KB of memory is required to hold pre-trained weights for the $(16, 8)$ NND proposed in [9]. It exceeds the capacity of the L1 and occupies most of the L2 cache in modern processors, and poses a substantial challenge for a hardware implementation as many of those weights have to be stored. Moreover, the floating-point arithmetic required is expensive in both hardware and energy consumption. This is a limiting factor for deployment of DL decoders in 5G networks [11].

The current state of art SCL decoders are able to decode polar codes using less than 8 bits [4], [6]. The research done in the field of machine learning (ML) proved that NNs do not require a high precision to achieve accuracy. [12], [13], [14] showed that NNs with quantized and compressed weights can achieve comparable results to the floating-point networks using only integer arithmetic. Some authors achieved a low error with as little as 1 bit used for training and inference [15]. Recently a RNN-BP decoder with quantized coefficients was proposed [11] proving that a low precision decoding is applicable in the domain of NN-based polar decoders. Moreover authors in [16] laid foundations for the hardware implementation of NN-based communication algorithms with compelling results.

This paper makes the following contributions:

- We present a new, simplified NN structure of the NND that does not impact the theoretical wireless performance.
- We present a new low-precision fixed-point decoder that reduces weights' size up to 16 times with wireless performance degradation below 0.4dB.
- We present an improved shift based matrix multiplication for the most compact, 2-bit decoder.
- We investigate how NN quantization impacts the wireless performance for different NN configurations and show that a total up to 54 times memory saving can be achieved.
- We show that up to 438 times and 555 times reduction in L1 and L2 data cache misses respectively can be achieved in the prototype software implementation on the general purpose processor.

In our low-precision NND, the reduced weight size results in improved cache behavior, with the smallest configuration allowing to fit all data into the L1 cache. This reduces the energy consumption and increases performance as the number of off-chip data accesses is reduced. Using 8-bit arithmetic also allows more data to be processed simultaneously using vector instructions in CPUs. Finally, this also leads to more efficient hardware implementations. As our low-precision NND uses integer-only arithmetic, with small data sizes, it can be implemented in hardware with lower space and memory requirements, allowing to exploit more parallelism and reducing the energy consumption.

The rest of the paper is organized as follows. In section II we briefly introduce polar codes and NN decoding. In section III we describe the experimental setup. In sections IV we describe the low-precision NND. In section V we present experimental results. And finally in section VI we draw some conclusions.

II. Background

A. Polar Codes

A polar code (N, K) is constructed by applying the transformation matrix $\boldsymbol{G_N}$ to the bit sequence $\boldsymbol{u} = \{u_0, u_1, ..., u_{N-1}\}$ as $\boldsymbol{x} = \boldsymbol{u}\boldsymbol{G_N}$, where N is the length of the codeword and K is the number of message bits. The bit sequence $\boldsymbol{u}$ is created by inserting K message bits into K most reliable bits positions in the vector. Remaining $N - K$ bits, referred to as frozen bits, are set to the fixed value and are known to both the decoder and the encoder. The transformation matrix $\boldsymbol{G_N} = \boldsymbol{G}^{\otimes n}$ is calculated as a n-th Kronecker power of the generator matrix $\boldsymbol{G}$, where $n = log_2(N)$.

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{1}$$

B. Neural Network Decoder

The NND is a fully connected NN that contains M hidden layers with non-zero sizes of $(l_1, l_2, ..., l_M)$. The size of the input and output of the network is set to be the length N of the codeword of the polar code $(N,\ K)$. We denote the full network structure as $(l_0, l_1, l_2, ..., l_M, l_{M+1}) = (N, l_1, l_2, ..., l_M, N)$.

The output of a layer m is calculated as:

$$\boldsymbol{y_m} = f_m(\boldsymbol{y_{m-1}}) = \alpha_m(\boldsymbol{y_{m-1}} \cdot \boldsymbol{K_m} + \boldsymbol{b_m}) \tag{2}$$

where $0 < m \leq (M+1)$, $\boldsymbol{x} = \boldsymbol{y_0}$ is the input vector of log-likelihood ratio (LLR) values and α_m is the activation function. $\boldsymbol{K_m}$ and $\boldsymbol{b_m}$ are respectively the weights' matrix and the bias associated with the layer m such that $\boldsymbol{K_m}$ is the matrix of size $l_{m-1} \times l_m$ and $\boldsymbol{b_m}$ is the vector of size l_m. Both the weights and bias are derived during the NN training process. Vector y_{M+1} is the output of the decoder with values that can be interpreted as probabilities of the output bits being 1.

We set the activation α_m function to be a rectified linear unit (3) for $0 < m \leq M$ and the sigmoid function (4) for $m = (M+1)$. The sigmoid function restricts the range of the decoder's output values to $[0, 1]$.

$$ReLU(x) = max(0, x) \tag{3}$$

$$S(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

The entire decoder can be written as:

$$\boldsymbol{y} = f(\boldsymbol{x}) = f_{M+1}(f_M(...f_1(\boldsymbol{x}))) \tag{5}$$

Estimated bit values of the sent messages $\hat{\boldsymbol{u}}$ are calculated from $\boldsymbol{y}$ as:

$$\hat{u}_i = \begin{cases} 0 & when\ p_i < 0.5 \\ 1 & when\ p_i \geq 0.5 \end{cases} \tag{6}$$

where p_i is the $i-th$ element of the vector $\boldsymbol{y}$.

III. NND Training Setup

We use TensorFlow (TF) [17] with Keras [18], as a convenient level of abstraction, with settings based on [7][9]. We train the NN with 3 hidden layers to decode $(16, 8)$ polar code using Adam optimizer [19], Mean Squared Error (MSE) loss function and batch size of 256. In every epoch, the network is fed with LLR values calculated for all possible codewords in 1dB signal-to-noise ratio (SNR) for total of 2^{16} epochs. We configure the decoder to output all 16 bits (frozen and message bits), rather than estimating only 8 message bits, so it can be easier integrated into the NSC decoder.

To test the decoder[1], a simulation is run for 6 SNR values in the range of 1dB to 6dB in the Additive White Gaussian Noise (AWGN) channel with Binary Shift-Phase Keying (BPSK) modulation. For each SNR value the decoder is run 100,000 times and the Block Error Probability (BLER) is calculated. We decided to use BLER, rather than Bit Error Probability (BER), as a metric as it is more useful in the context of a real system.

IV. Low-Precision Neural Network

A. Activation Functions

Whereas the ReLU function is easy to compute, the sigmoid function uses the expensive exponent function. Keras allows replacing this with a simplified, hard sigmoid function (7) that only requires basic multiplication, addition and comparison operations.

$$S(x) = \begin{cases} 0 & when\ x < -2.5 \\ 1 & when\ x > 2.5 \\ 0.2 \cdot x + 0.5 & otherwise \end{cases} \tag{7}$$

Fig. 1 shows that using the hard sigmoid function has no substantial impact on the BLER performance of the decoder. Because of that the simplified function is used in the low-precision decoder.

[1]To make results reproducible, we have made the source code freely available: https://github.com/IgWod/low-precision-nnd

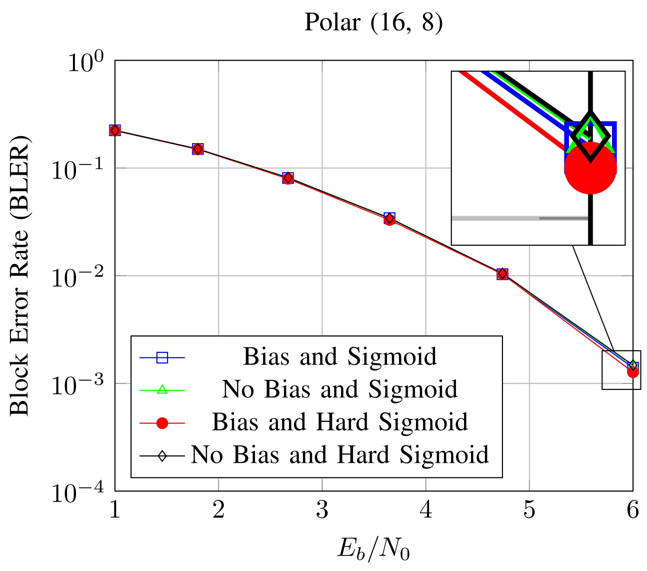

Fig. 1. BLER performance comparison of the floating-point NND (512, 256, 128) trained in four different configuration.

B. Bias

While bias is essential for successfully training certain NNs, there is no BLER performance degradation when bias is removed from the training of the NND. The result of using the unbiased NN can be seen on Fig. 1. Removing the bias from the network not only reduces computational complexity and storage requirements, but also prevents extra quantization error being introduced by bias addition. Due to those facts we do not use bias in the NND.

C. Weights Quantization

Although removing the bias and simplifying the activation function decreases the complexity of the NND, the main limiting factor is the memory required to store NN's weights. Moreover, wide data types increase the energy consumption and may limit the amount of data that can be processed in parallel. The described problems can be alleviated by quantizing weights with a limited number of bits.

We perform all of the arithmetic and quantize the input and output with 8 bits in Q8.4 fixed-point number format. From our observations, using smaller number of bits leads to a significant BLER performance degradation as saturation becomes a problem especially in the high SNR region. However weights can be stored on 8 bits or less in the fixed-point data format that uses $n-1$ for the fractional part and 1 bit for the sign, where n is the number of bits used for quantization.

To reduce the impact of the post-training quantization on BLER performance we use a quantization aware training, so that the quantization error can be incorporated into the training loss and the NN learns how to perform decoding with a limited precision. As TF does not support a native training on less than 8 bits, we use the fake quantization function to introduce the quantization error into the NN. With the fake quantization the precision of data is reduced, however data is still stored in floating-point during the training. Due to

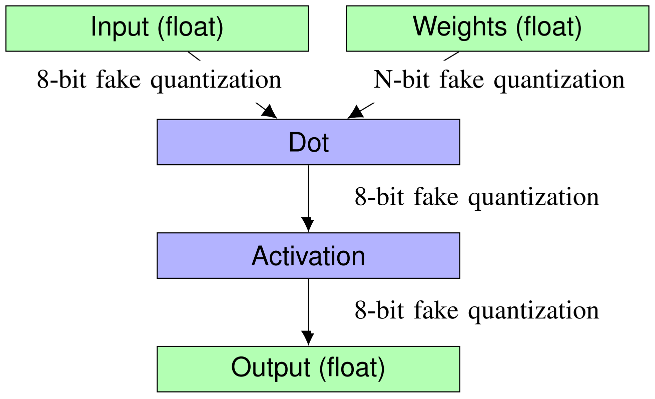

Fig. 2. Data flow in the layer during the quantization aware training. The fake quantization is performed on the edges of the flow graph.

that all operations during the training are performed using the floating-point arithmetic. The quantization process can be seen on Fig. 2. The quantization aware training results in around 30% longer training time, compared to the initial NN.

Results of our experiments showed that the direct quantization with less than 4 bits caused the wireless performance to be no longer acceptable. However from the histogram of the weights quantized with 4 bits (Fig. 3) we can see that most of the values fall within the range $[-0.25, 0.125]$ and those values cannot be represented with less than 4 bits. Based on that we can encode weights with 2 bits by assigning to each of the 2-bit values one of the four most frequent weights and further reduce the space requirement to up to 2 bits per single weight value. To provide a mapping between 2-bit values and the actual weights small lookup table (LUT) can be used. It introduces an additional degradation into the BLER performance in exchanged for further reduction in the space requirement. Because small LUT fits into the L1 cache or modern CPU registers, no extra memory accesses are required to perform the mapping. To incorporate a more strict quantization into the training process we clip the weights' matrix values in range $[-0.25, 0.125]$ after quantization.

D. Matrix multiplication

An additional optimization can be done in the implementation of the compact 2-bit decoder. We observe that all weights remaining after quantization and clipping are either 0 or the power of 2. As multiplication by powers of 2 can be implemented as a shift operation, the direct multiplication in the matrix multiplication in the equation (2) can be eliminated. The left shift ($x << n$) acts on the binary representation of x by shifting bits n times towards the most significant bit, e.g. 00010100 << 2 becomes 01010000. The right shift is defined symmetrically. Assuming a and b use the same number of bits to represent the fractional part and both are positive[2] the fixed-point multiplication can be performed as follows:

$$((a \cdot b) >> n) + 1) >> 1 \tag{8}$$

[2]The multiplication can be defined for negative numbers in a similar way, but we omit it here to simplify the description.

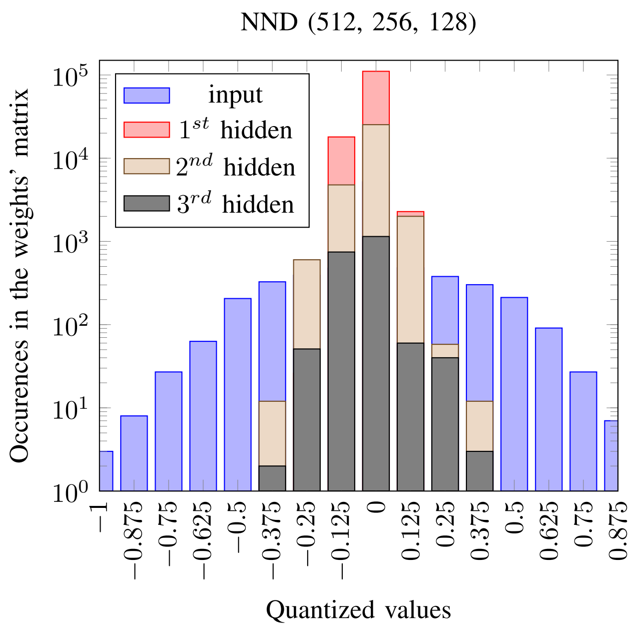

Fig. 3. Distribution of weights values for every layer quantized with 4 bits.

Where $\cdot$ is the standard integer multiplication, shifts and addition are used to scale and round the result and $n = \#frac\_bits - 1$. Instead of mapping 2-bit values into 4-bit fixed-point numbers we can use a LUT to map those values into appropriate shifts, so we can remove the multiplication:

$$((a << s) >> n) + 1) >> 1 \tag{9}$$

Where s is the shift read from the LUT. If $s < n$ then:

$$((a >> (n - s)) + 1) >> 1 \tag{10}$$

We can store $(n-s)$ in the LUT to avoid performing subtraction. As shown the matrix multiplication can be implemented using only shifts and addition, and an additional simple logic handles positive and negative values.

V. Experimental Results

A. BLER Performance

Fig. 4 shows how the quantization with different number of bits impacts the BLER performance. As it can be seen the wireless performance degradation for 8-, 4- and 2-bit quantization is respectively around 0.1dB, 0.2db and 0.4dB. However low precision decoding leads to up to 16 times saving in the memory size required to store weights. The increased degradation in the high SNR region is caused by the saturation of 8-bit values and it can be mitigated by increasing the number of bits used to perform calculations. The potential saving from the quantization with specific number of bits can be seen in the Table I.

TABLE I
Weights' size for different quantization schemes

Type	float	int8	int4	lut2
Size (bytes)	696 320 (100%)	174 080 (25%)	87 040 (12.5%)	43 520 (6.25%)

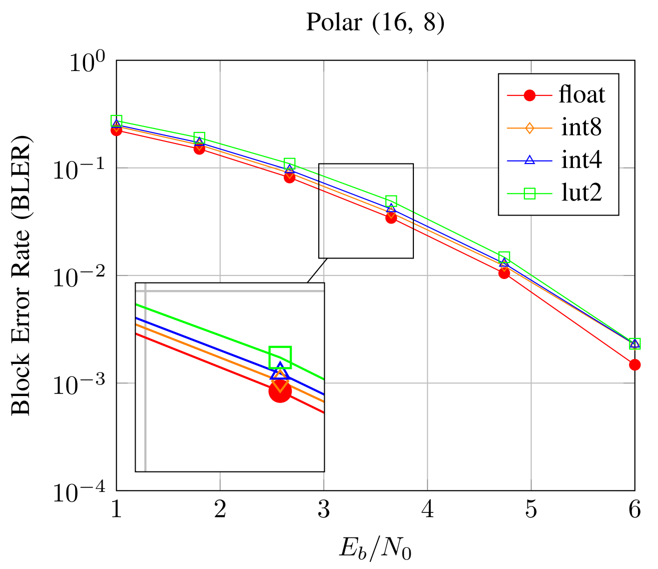

Fig. 4. BLER performance comparison of the NND (512, 256, 128) with quantized weights.

TABLE II
Weights' size for different quantization schemes and configurations

NN Configuration	(512, 256, 128)	(512, 256, 128)	(256, 128, 64)	(128, 64, 32)
Type	float	int8	int4	int8
Size (bytes)	696 320 (100%)	174 080 (25%)	23 040 (3.3%)	12 800 (1.8%)

We also investigated how sizes of the layers in the NND affect the BLER performance of the decoder with quantized weights (Fig. 5). It can be seen that the $(256, 128, 64)$ NND quantized with 4 bits has a 0.1dB degradation compared to the $(512, 256, 128)$ 8-bit decoder. Moreover $(128, 64, 32)$ 8-bit decoder has the wireless performance comparable with the $(512, 256, 128)$ 2-bit decoder. All other tested decoders have a BLER performance degradation larger than 0.5dB. The penitential memory saving can be see in the Table II.

Further reduction in the space requirements could be achieved by compressing zero weights, as they account for the most of the values stored in the memory.

B. Cache efficiency

To show the impact of the reduction in the weights' size on the cache efficiency we developed a prototype software implementation of the NND. We use performance application programming interface (PAPI) [20] to access the hardware performance counters on the Intel i5-6200U processor (32KB L1d, 32KB L1i and 256KB L2 cache). We measured L1 and L2 data cache misses for 100 execution of the decoder with warm cache repeating the experiment 2500 times.

As it can be seen on Fig. 6 changing the data type from float to int8 yields around 4 times improvement in the number of L1 data cache misses and 9 times in the number of L2 cache misses. Reducing each layer's size by the factor of 2 and 4 results in respectively around 15 and 438 times cache misses

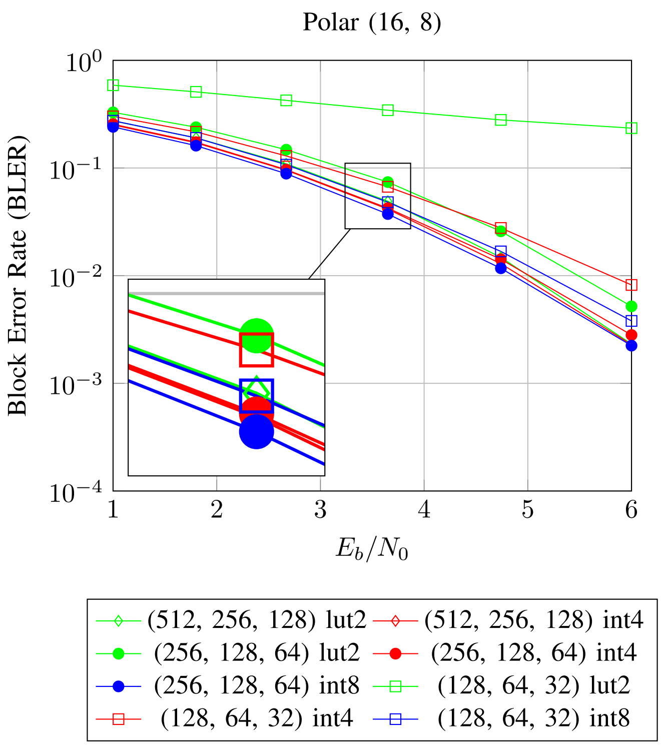

Fig. 5. BLER performance comparison of different NNDs' configurations.

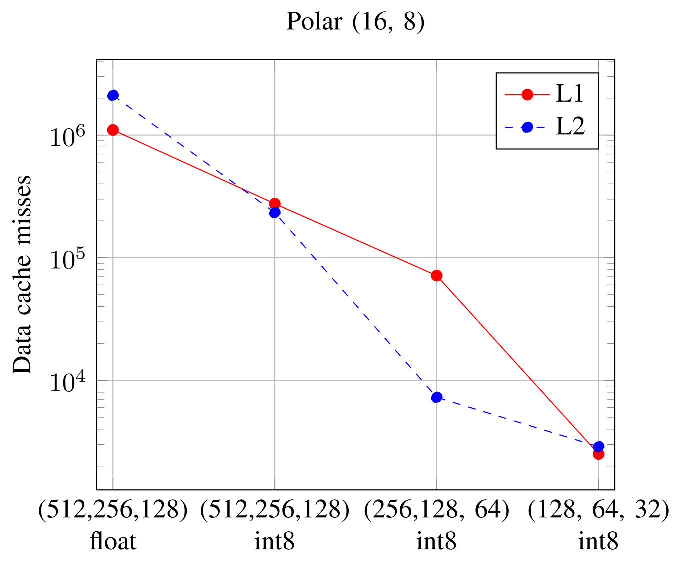

Fig. 6. Mean number of data cache misses for 100 executions of the decoder in different neural network configurations.

reduction in L1 and 290 and 555 times in L2 compared to the full floating-point decoder.

VI. Conclusion

We presented a new, simplified, low-precision NND that alleviates the high memory requirement of the floating-point decoder. Our technique reduces the weights' size to up to 54 times and the number of data cache misses up to 555 times, while wireless performance degradation remains below 0.4dB. In future work, we will integrate the low-precision NND with the NSC decoder to enable decoding of large codewords required in 5G NR. Moreover it was recently showed [21] that the NN structure is applicable for the decoding of LDPC codes, where we expect our technique is likely to yield similar benefits with minor degradation.

References

[1] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes," in *2008 IEEE International Symposium on Information Theory*. IEEE, 2008, pp. 1173–1177.

[2] "Final report of 3GPP TSG RAN WG1 #87 v1.0.0, Reno, USA," November 2016.

[3] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[4] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE transactions on signal processing*, vol. 63, no. 19, pp. 5165–5179, 2015.

[5] S. A. Hashemi, C. Condo, and W. J. Gross, "Fast and flexible successive-cancellation list decoders for polar codes," *IEEE Transactions on Signal Processing*, vol. 65, no. 21, pp. 5756–5769, 2017.

[6] S. A. Hashemi, C. Condo, F. Ercan, and W. J. Gross, "Memory-efficient polar decoders," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 7, no. 4, pp. 604–615, 2017.

[7] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2017, pp. 1–6.

[8] S. Cammerer, T. Gruber, J. Hoydis, and S. ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[9] N. Doan, S. A. Hashemi, and W. J. Gross, "Neural successive cancellation decoding of polar codes," in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2018, pp. 1–5.

[10] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE communications letters*, vol. 15, no. 12, pp. 1378–1380, 2011.

[11] C.-F. Teng, C.-H. Wu, K.-S. Ho, and A.-Y. Wu, "Low-complexity recurrent neural network-based polar decoder with weight quantization mechanism," *arXiv preprint arXiv:1810.12154*, 2018.

[12] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[14] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[15] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[16] F. A. Aoudia and J. Hoydis, "Towards hardware implementation of neural network-based communication algorithms," *arXiv preprint arXiv:1902.06939*, 2019.

[17] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[18] F. Chollet *et al.*, "Keras," 2015.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with PAPI-C," in *Tools for High Performance Computing 2009*. Springer, 2010, pp. 157–173.

[21] Y. Wang, Z. Zhang, S. Zhang, S. Cao, and S. Xu, "A unified deep learning based polar-LDPC decoder for 5G communication systems," in *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2018, pp. 1–6.