Sparrow ECC: A Lightweight ECC Approach for HBM Refresh Reduction towards Energy-efficient DNN Inference

Hoseok Kim[†], Seung Hun Choi[†], Young-Ho Gong[‡], Joonho Kong[§], and Sung Woo Chung[†]
[†]Department of Computer Science and Engineering, Korea University, Seoul, South Korea
[‡]School of Software, Soongsil University, Seoul, South Korea
[§]School of Electronic and Electrical Engineering, Kyungpook National University, Daegu, South Korea
{thomas1324, csh30096, swchung} @korea.ac.kr; yhgong@ssu.ac.kr; joonho.kong@knu.ac.kr

ABSTRACT

Exponential growth in deep neural network (DNN) model size has resulted in significant demands for memory bandwidth, leading to the extensive adoption of high bandwidth memory (HBM) in DNN inference. However, with the shorter retention time due to high operating temperature, HBM requires more frequent refresh operations, suffering larger refresh energy/performance overhead. In this paper, we propose Sparrow ECC, a lightweight but stronger HBM ECC technique for less refresh operations while preserving inference accuracy. Sparrow ECC exploits the dominant exponent pattern (i.e., value similarity) in pre-trained DNN weights, limiting the exponent value range of the pre-trained weights to prevent anomalously large weight value change due to the errors. In addition, through duplication and single error correction (SEC) code, Sparrow ECC strongly protects the critical bits in DNN weights. In our evaluation, when the proportion of 1⟶0 bit errors is 100% and 99%, Sparrow ECC reduces the refresh energy consumption by 90.40% and 93.22%, on average, respectively, compared to the state-of-the-art (RS(19,17)+ZEM [22]) refresh reduction technique, while preserving inference accuracy.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; • **Hardware** → *Error detection and error correction.*

KEYWORDS

Deep neural networks, DRAM refresh, ECC, Energy efficiency

ACM Reference format:

Hoseok Kim, Seung Hun Choi, Young-Ho Gong, Joonho Kong, and Sung Woo Chung. 2024. Sparrow ECC: A Lightweight ECC Approach for HBM Refresh Reduction towards Energy-efficient DNN Inference. In *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '24), August 5–7, 2024, Newport Beach, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3665314.3670825

1 INTRODUCTION

Deep neural networks (DNNs) have become prevalent solutions across diverse artificial intelligence (AI) domains such as computer vision and natural language processing. To achieve high accuracy, there has been exponential growth in DNN model size [3]. Unfortunately, larger models require high memory bandwidth to reduce the latency, which has high bandwidth memory (HBM) employed in DNN inference [10][33]. However, DRAMs including HBM have substantial refresh energy/performance overhead [6][22]; HBM as well as DDR5 has a 32 ms refresh period to prevent retention errors. Such periodic refresh operations not only stall other memory requests but also consume significant energy. As DRAM process technology scales down, the retention time of DRAM cells decreases [17], exacerbating the refresh overhead due to more frequent refresh operations. Furthermore, HBM refresh overhead becomes more severe, due to 1) high capacity and 2) high operating temperature. First, higher capacity results in more refresh operations. Second, since HBM is typically integrated with high-performance (i.e., high power consuming) processing units, its temperature goes over 90 ℃ by the heat propagation from a hot processing unit through the silicon interposer [13]. Additionally, as the number of stacks increases in recent HBM, the 3D stacked structure leads to higher operating temperatures [12][23], causing capacitors to lose charge more quickly; the number of retention errors increases exponentially with temperature [6]. Therefore, to mitigate the increasing retention errors, recent HBM adopts a much shorter refresh period of up to 8 ms at high temperature as well as stronger error correction code (ECC) (i.e., Reed-Solomon (RS) code) than the conventional DRAMs [32].

In this paper, we propose Sparrow ECC, a lightweight ECC scheme to reduce the refresh overhead of HBM while preserving DNN inference accuracy. Sparrow ECC is applied to Bfloat16 DNN weights[1], considering the datatype that exhibits negligible accuracy loss as well as less computation overhead in datacenter inference; FP32 weights require significant computation overhead, while INT8 quantization leads to 5.98% accuracy loss [22], on average, which is unacceptable (i.e., >1%) in datacenter inference [1][3][10]. According to our analysis on pre-trained neural networks, the pre-trained weights have similarity in exponent values; we observed that the difference between the exponent values of 64 consecutive weights is generally less than 16 (or 32). With the maximum exponent value among 64 consecutive weights, Sparrow ECC replaces each exponent value of the weights with a delta value (i.e., difference between the maximum exponent value and its exponent value) with less bits. Exploiting the saved bits (i.e., remaining bits) for better error correctability, Sparrow ECC 1) applies Single Error Correction (SEC) code to the exponent delta value and 2) redundantly stores the maximum exponent and sign bits in the remaining saved bits. By redundantly storing the maximum exponent bits, Sparrow ECC prevents weights from being anomalously skewed due to uncorrectable retention errors. Sparrow ECC provides stronger error correctability with lightweight computation for parity check compared to the RS code of the off-

[1] In this paper, DNN weights refer to all DNN parameters including weights, biases, and embeddings.

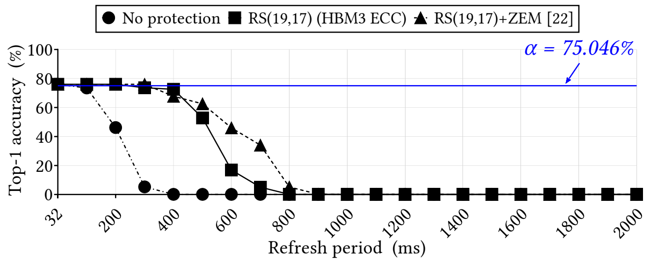


Figure 1: Top-1 accuracy of ResNet50 depending on refresh period at a 90 ℃ HBM. (*α indicates 1% accuracy loss*)

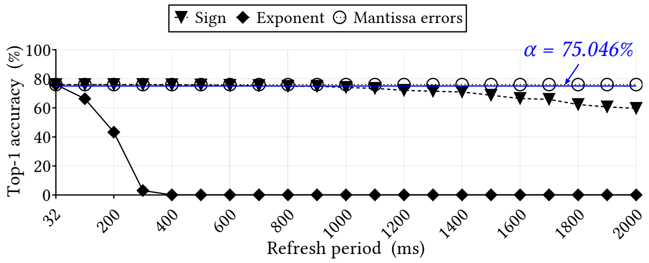


Figure 2: Top-1 accuracy of ResNet50 depending on refresh period when errors occur in each part at a 90 ℃ HBM with no protection. (*α indicates 1% accuracy loss*)

the-shelf HBM3, without additional storage overhead. Consequently, we significantly extend the refresh period of HBM with Sparrow ECC, achieving substantial reduction in refresh overhead with negligible accuracy loss in contemporary DNN models.

2 RELATED WORK

To reduce the refresh overhead, it is possible to extend the refresh period while correcting the retention errors by stronger ECC. Lee et al. proposed Stealth ECC [15], a stronger ECC scheme that stores more parity bits in the empty bits of narrow-width values (i.e., 64-bit values in which the upper 32-bit is 0). Bae et al. proposed Twin ECC [2], which also exploits narrow-width values, adopting a duplication based stronger ECC scheme. However, the ECC schemes are not strong enough for refresh reduction of HBM, especially in case of DNN inference due to the reasons as follows; 1) narrow-width values are mostly 32-bit integer, whereas DNN weights are generally floating point and 2) the schemes protect all data bits (except the empty bits) with equal importance. The importance of bits in DNN weights varies; a bit error in the sign or exponent bit has a much greater impact on accuracy than the mantissa bit.

Due to the large memory footprint of DNN weights, there have been ECC schemes dedicated for DNN weights to reduce the refresh overhead. Nguyen et al. proposed St-DRC [21], which applies SEC code to the 9-bit (= 1-bit sign + 8-bit exponent) of DNN weights. Nguyen et al. also proposed ZEM [22], which decreases the number of '1' bits in the exponent bits of DNN weights, since errors in true-cell DRAM are mostly 1⟶0 bit errors. However, since the ECC schemes [21][22] are proposed for DRAM and HBM which operate at 60 ℃, they are not strong enough to operate at 90 ℃. In other words, though they reduce the refresh overhead, they are not capable of correcting retention errors at the high operating temperature (i.e., 90 ℃).

3 BACKGROUND AND MOTIVATION

Most of the cutting-edge neural network applications have been running on high-performance GPUs with HBM to store/serve a considerable number of weights with sufficient memory bandwidth. If retention errors occur in the HBM, they compromise inference accuracy. However, in data centers, the acceptable accuracy loss is less than 1% for image classification models and less than 0.1% for natural language processing models [1][3][10].

To analyze the impact of retention errors on accuracy at 90℃ HBM [13][19] when exploiting existing ECC schemes, we evaluate the top-1 accuracy of a representative image classification model (ResNet50 [7]) depending on refresh period. We determine retention error rate depending on refresh period based on the data provided in [19]. We measure the accuracy of the following three schemes; 1) no protection, 2) the off-the-shelf HBM3 ECC scheme (RS(19,17) [32]), and 3) the previously proposed scheme (ZEM [22]). Since ZEM does not utilize the parity bits for ECC, it is integrated with RS(19,17) for a fair comparison. Considering the dominant retention errors (i.e., 1⟶0 bit errors) due to the extended refresh period (i.e., >32 ms refresh period) over the other error types (i.e., disturbance errors and radiation induced errors), we set the proportion of 1⟶0 bit errors as 99%.

As shown in Figure 1, in case of no protection, the model results in unacceptable accuracy loss (i.e., accuracy drops below α) when the refresh period is longer than 100 ms. With no protection, the errors cause large weight value change, resulting in the rapid deterioration in accuracy. In case of RS(19,17) and RS(19,17)+ZEM, the model shows unacceptable accuracy loss when the refresh period is longer than 300 ms. Due to the overlooked high temperature of HBM, at extended refresh periods, the ECC schemes cannot correct the exponentially increased retention errors, which leads to unacceptable accuracy loss. Thus, when RS(19,17) and RS(19,17)+ZEM extend the refresh period of typical DRAM modules such as DIMMs from 32 ms to 512 ms or over 3000 ms [21][22], they are not applicable to a 90 ℃ HBM.

To design a robust and lightweight ECC scheme for DNN weights, we determine the important bits in the weight element that significantly affect DNN accuracy. We measure the top-1 accuracy of ResNet50 when the retention error(s) occurs in each part (the sign, exponent, and mantissa bits) of the DNN weights with no protection. As shown in Figure 2, errors in the exponent bits easily result in unacceptable accuracy loss, which is due to the anomalously large weight value change caused by the errors. In case of the 8-bit exponent, a single bit error increases/decreases the weight by up to 2^{128}X of the original weight value, resulting in unacceptable accuracy loss when the refresh period is longer than 32 ms. For the 1-bit sign, a bit error increases/decreases the weight by 2X of the absolute value, leading to unacceptable accuracy loss when the refresh period is longer than 800 ms. On the other hand, errors in the mantissa bits have a much lower impact on accuracy. Therefore, it is more important to protect the exponent and sign bits, rather than mantissa bits in the DNN weights.

4 SPARROW ECC: LIGHTWEIGHT ECC FOR DNN INFERENCE

4.1 Overview

The memory access granularity of Sparrow ECC is 1024-bit, considering the LLC line size of a typical GPU [33]. On a memory write operation (encoding), Sparrow ECC selects the ECC scheme depending on whether the 1024-bit data consists of DNN weights or not. To support ECC selection depending on weight/non-weight block, we add weight read/write commands to the column commands of HBM [32]; since the HBM standard exploits only 6 column commands, we can add the new commands exploiting the unused column commands, with minor modifications in the column

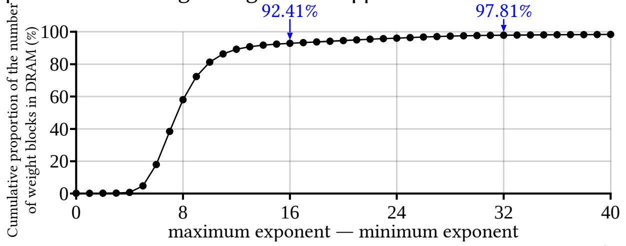


Figure 3: *maximum exponent − minimum exponent* value distribution of weight blocks, average of 78 pre-trained models.

decoder. Thus, when the GPU reads or writes data from/to HBM, the memory controller generates control signals for the multiplexer/demultiplexer depending on the column command to select either Sparrow ECC encoder or RS(19,17) encoder (off-the-shelf HBM3 ECC scheme [32]). When the 1024-bit data consists of DNN weights, the Sparrow ECC encoder takes 64 consecutive DNN weights (1024-bit), which we call a *weight block*, as input. The encoder applies Sparrow ECC to the weight block, generating the 1152-bit (= 1024-bit weights + 128-bit parity) encoded weight block, which is written to the DRAM cells. Since our Sparrow ECC generates the same number of parity bits as the JEDEC HBM3 standard ECC (RS(19,17)), it does not require additional storage overhead compared to the off-the-shelf HBM3 [32]; as the implementation of additional metadata is vendor specific (optional), we do not consider it in this paper. On the other hand, when the 1024-bit data is a non-weight block (e.g., GPU kernel code), Sparrow ECC adopts the RS(19,17), which is the off-the-shelf HBM3 ECC scheme [32]. The RS(19,17) encoder generates a 1152-bit encoded non-weight block.

4.2 Operation of Sparrow ECC

For a weight block, the Sparrow ECC encoder prevents the anomalously large weight value change by limiting the value of the 8-bit exponent to a specific range with the maximum exponent and *delta* value (i.e., *maximum exponent − exponent* value). In our motivational experiment, we analyzed the weight value differences for 78 widely-used DNN models including transformer-based as well as CNN-based models. As shown in Figure 3, on average of 78 DNN models, 92.41% (or 97.81%) of the *maximum exponent − minimum exponent* values in each weight block are below 16 (or 32), which means that most weight values are concentrated in a narrow range. Furthermore, the state-of-the-art transformer-based models have narrower weight distribution (i.e., smaller weight value differences) than the conventional CNN-based models; for example, in recent transformer-based large language models (LLMs), ~99.9% of the weight values are concentrated in the range of [-0.1, 0.1] [8][11], while ResNet50 weight values are mostly in the range of [-0.2, 0.2] [9]. Therefore, we can suppose that most *maximum exponent − minimum exponent* values of weight blocks in recent DNN models are also less than 16 (or 32). Based on the observation, the maximum exponent is the upper bound of the exponent in a weight block and the *maximum exponent* − 15 (*or* 31) is the lower bound.

Figure 4 depicts the encoding process of the Sparrow ECC encoder. In each weight block, to distinguish the two types (*maximum exponent* − 15 or *maximum exponent* − 31) of lower bound, the Sparrow ECC encoder employs a 1-bit flag. The Sparrow ECC encoder then redundantly stores the 8-bit maximum exponent (upper bound) and 1-bit flag in the weight block seven times, exploiting 63-bit (= 7 * 8-bit maximum

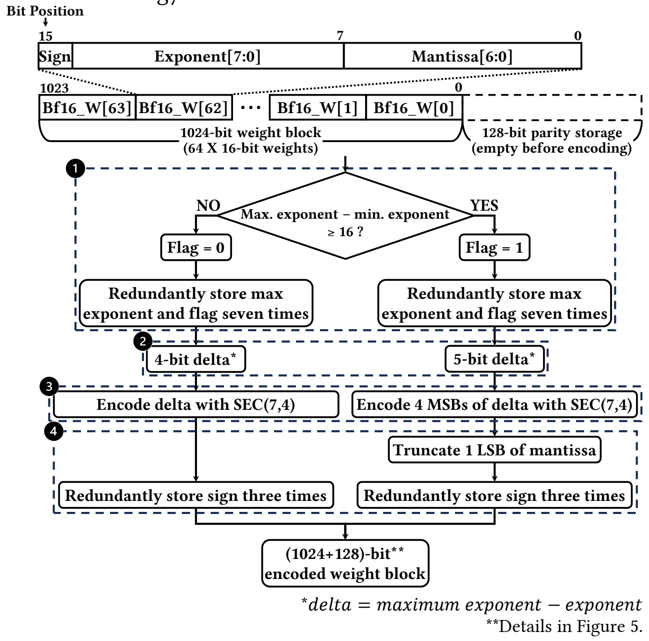

Bit Position
15
7
0
Sign
Exponent[7:0]
Mantissa[6:0]
1023
0
Bf16_W[63]
Bf16_W[62]
Bf16_W[1]
Bf16_W[0]
1024-bit weight block
(64 X 16-bit weights)
128-bit parity storage
(empty before encoding)
1
Max. exponent − min. exponent ≥ 16 ?
NO
YES
Flag = 0
Flag = 1
Redundantly store max exponent and flag seven times
Redundantly store max exponent and flag seven times
2
4-bit delta*
5-bit delta*
3
Encode delta with SEC(7,4)
Encode 4 MSBs of delta with SEC(7,4)
4
Truncate 1 LSB of mantissa
Redundantly store sign three times
Redundantly store sign three times
(1024+128)-bit**
encoded weight block
*delta = maximum exponent − exponent
**Details in Figure 5.

Figure 4: Encoding process of the Sparrow ECC encoder.

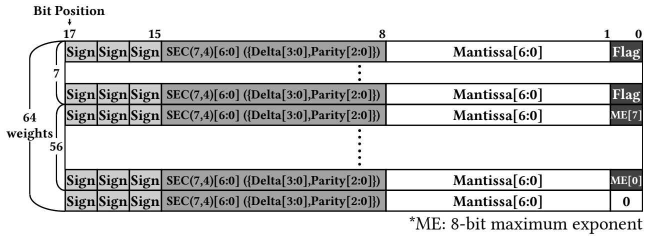

Bit Position
17
15
8
1
0
Sign
SEC(7,4)[6:0] ({Delta[3:0],Parity[2:0]})
Mantissa[6:0]
Flag
7
64 weights
56
ME[7]
ME[0]
0
*ME: 8-bit maximum exponent

Figure 5: (1024+128)-bit encoded weight block for 4-bit delta

exponent + 7 * 1-bit flag) of the 128-bit parity, thereby saving 65-bit parity (❶). With the seven replicas, Sparrow ECC achieves up to 3-bit error correctability for each bit position in the 8-bit maximum exponent and 1-bit flag by majority voting.

Since the maximum exponent is stored in the weight block, the delta in each weight only has to store a 4-bit or 5-bit delta (❷) instead of the 8-bit exponent. When the delta value is greater than 32, it is changed to 31 (5'b11111)[2]. Since the Sparrow ECC encoder replaces the 8-bit exponent in each DNN weight with 4-bit or 5-bit delta, the Sparrow ECC encoder saves 4-bit or 3-bit in each weight. Thus, a weight block with 64 DNN weights has 321-bit (= 64 * 4-bit saved bits + 65-bit parity) and 257-bit (= 64 * 3-bit saved bits + 65-bit parity) free space for 4-bit and 5-bit delta, respectively; saved 65-bit parity is explained in the previous paragraph.

To protect the 4-bit or 5-bit delta, exploiting the remaining saved bits, the Sparrow ECC encoder adopts SEC(7,4) encoding to the delta in each DNN weight (❸). In case of 4-bit delta, it is protected by SEC(7,4), whereas for 5-bit delta, the 4 MSBs are protected by SEC(7,4), since the MSBs are more sensitive to bit errors than the LSBs. Through SEC(7,4) encoding, the Sparrow ECC encoder utilizes 192-bit (= 64 * 3-bit parity) saved bits in the weight block, leaving 129-bit (= 321-bit − 192-bit) and 65-bit (=

[2] 31 is the largest value representable with a 5-bit; according to our experiments, limiting the delta value to 31 degrades the accuracy by only 0.06%, on average, since the number of weights with delta values larger than 31 is very small.

257-bit − 192-bit) free space for 4-bit and 5-bit delta, respectively.

Since an error in the 1-bit sign also causes accuracy loss, the Sparrow ECC encoder protects the 1-bit sign of each DNN weight by redundantly storing it three times, achieving 1-bit error correctability (❹). As shown in Figure 5, in case of 4-bit delta, the Sparrow ECC encoder additionally stores the 1-bit sign twice in 128-bit (= 64 * 2 * 1-bit sign) of the remaining 129-bit free space. However, in case of 5-bit delta, the weight block lacks 63-bit to additionally store the 1-bit sign. In this case, the Sparrow ECC encoder truncates 1 LSB of the 7-bit mantissa in each weight to generate 64-bit (= 64 * 1-bit) free space, since truncating 1-bit of the mantissa has a negligible impact on accuracy [3]. Then, the 1-bit sign is additionally stored in the 128-bit free space.

In case of the non-weight block (e.g., GPU kernel code), Sparrow ECC adopts the RS(19,17) encoder of the off-the-shelf HBM3. The RS(19,17) encoder takes 256-bit data and 16-bit metadata as input, generating 32-bit parity. To match the 1024-bit memory access granularity of Sparrow ECC, the RS(19,17) encoder encodes 256-bit data four times, generating the 1152-bit (= 1024-bit data + 128-bit parity) encoded non-weight block. Note the Sparrow ECC encoder has no storage overhead compared to the conventional RS(19,17) encoder, since it generates the same number of parity bits (128-bit) for 1024-bit data.

4.3 Refresh Control of HBM Channels

Since HBM consists of multiple independent channels, it is possible to distribute the memory address space to channels. Based on channel interleaving of HBM, when the memory controller loads the pre-trained weights from host (i.e., system memory) to GPU memory, the weight blocks can be allocated to specific channels separately, while the data other than weights is allocated to the other channels. By separating weight and non-weight blocks in different channels, we are able to apply Sparrow ECC to the HBM channels storing only weight blocks.

More importantly, exploiting the stronger error correctability of Sparrow ECC, we extend the refresh period of weight storing HBM channels, while tolerating the increased retention errors. Non-weight storing channels still operate with the conventional 32 ms refresh period. To determine the extended refresh period for weight storing channels, we find the longest refresh period that the DNN model shows acceptable accuracy loss by gradually extending the refresh period. Since each HBM channel operates independently, it takes commands from the memory controller independently [32]. Thus, it is possible to only change the refresh period of weight storing channels to the extended refresh period. Considering the large memory footprint of DNN weights [22], it would be possible for most of the HBM channels to adopt Sparrow ECC, thus adopting the extended refresh period.

5 EVALUATION

5.1 Evaluation Methodology

We evaluate Sparrow ECC in terms of refresh energy, average memory read latency, and area/power overhead, compared to the following three schemes; 1) no protection, 2) RS(19,17) (HBM3 ECC [32]), and 3) RS(19,17)+ZEM [22]. According to [31], most of the memory accesses during DNN inference are memory read requests. Furthermore, the ECC decoding latency is on the critical path of memory read requests. Thus, we consider the memory read latency as our performance metric. To analyze the refresh energy and average memory read latency, we implement the schemes on DRAMsim3 [16] using HBM3 parameters [32]. We consider a 16-channel HBM3 operating at 90 ℃, since HBM3 typically operates at high temperature [13]. Each channel has a capacity of 8 Gb.

Table 1. DNN models used in our evaluations.

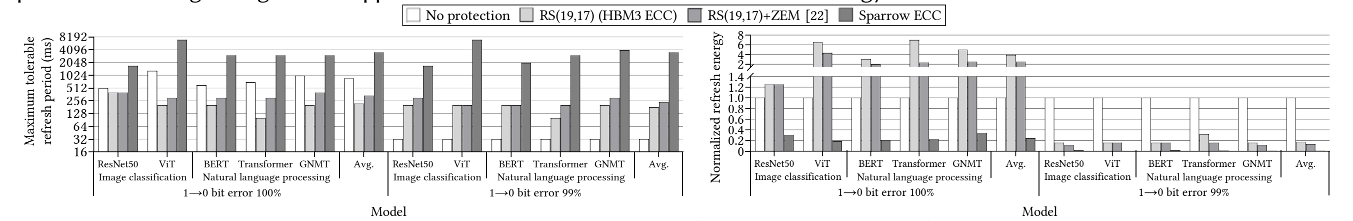
Model	Dataset	Accuracy	Model Size
ResNet50	ILSVRC2012	76.05%	51.20 MB
ViT-B/16	ILSVRC2012	81.04%	173.20 MB
BERT-Large	SQUAD 1.1	91.45 (F1)	590.20 MB
Transformer	wmt14 en-de	27.70 (BLEU)	1341.44 MB
GNMT	wmt16 en-de	24.36 (BLEU)	1536.00 MB

As described in Table 1, we use 5 DNN models implemented with PyTorch [25]; ResNet50 [7], ViT (Vision Transformer) [5], BERT [4], Transformer [29], and GNMT [30]. While running inference on the pre-trained DNN models shown in Table 1, we capture the memory access trace of each DNN model using Pin [18] for energy/performance simulation of HBM3 with DRAMsim3 [16].

To evaluate the refresh energy, we examine the maximum tolerable refresh period for each scheme using the uniform random bit error injection method [22]. Note the maximum tolerable refresh period is the longest refresh period by which a DNN model shows acceptable accuracy loss (i.e., 1% for image classification and 0.1% for natural language processing [1][3][10]). We determine the number of retention errors depending on refresh period by the data provided in [19]. In [2], the proportion of 1⟶0 bit errors is 80%, 90%, and 100% at 60 ℃ DRAM. In [19], the number of retention errors increases by more than 100X when DRAM operating temperature increases from 60 ℃ to 90 ℃. So, we can find out the 1⟶0 bit error proportion is larger than 99.75% (=(100*80%)/(100*80%+20%)) when the temperature increases from 60 ℃ to 90 ℃. Thus, we evaluate the schemes with 1⟶0 bit error proportion of 100% and 99%.

We measure the average accuracy of 30 iterations of inference for each DNN model to prevent the over/under estimation of accuracy due to the randomness of bit error injection. By reflecting the maximum tolerable refresh period into DRAMsim3 [16], we evaluate the refresh energy of each scheme.

To evaluate the average memory read latency, we run 10 iterations of inference on each DNN model with different inputs. In addition, we reflect the encoding and decoding latencies of the three ECC schemes to DRAMsim3 [16]. For RS(19,17), since the hardware implementation is undisclosed by DRAM manufacturers, we use the latency results from [26]. According to [26], when synthesized using a 45 nm library, the encoding and decoding latencies of RS(19,17) are 1 cycle and 8 cycles, respectively, in an HBM3 operating at 1.6 GHz. ZEM [22] adds 1 XOR gate and 1 inverter to the encoding and decoding process. Thus, we consider the encoding and decoding latencies of RS(19,17)+ZEM to be the same as RS(19,17). For a fair comparison, we use the FreePDK 45 nm library [28] to synthesize the encoding and decoding logic of Sparrow ECC at 1.6 GHz with the Synopsys Design Compiler. The synthesis results show that the encoding and decoding latencies of Sparrow ECC are 4 cycles and 1 cycle, respectively. The longer encoding latency of Sparrow ECC is due to the comparison operation (Figure 4 ❶) required to determine the exponent range, while the decoding latency is much shorter compared to RS(19,17)



(a) Maximum tolerable refresh period (accepting accuracy loss <1% or <0.1%, depending on model) (b) Normalized refresh energy

Figure 6: Maximum tolerable refresh period/normalized refresh energy depending on ECC schemes across DNN models.

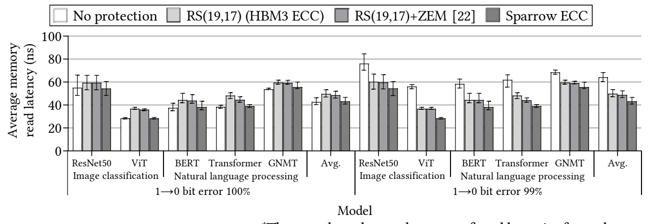
due to the simple majority voting and SEC(7,4) decoding.

5.2 Refresh Overhead Reduction

Figure 6(a) and (b) illustrate the maximum tolerable refresh period and normalized refresh energy, respectively, depending on the schemes across 5 DNN models. As shown in Figure 6(a), when the proportion of 1⟶0 bit errors are 100% and 99%, Sparrow ECC achieves a significantly long maximum tolerable refresh period of 3540 ms, on average, for both proportions. Sparrow ECC provides strong error correctability regardless of the proportion of 1⟶0 bit errors, since the majority voting and SEC(7,4) protection is applicable to 0⟶1 as well as 1⟶0 bit errors. In addition, Sparrow ECC prevents anomalously large weight value change by the delta, which further improves the robustness of DNN weights.

In contrast, RS(19,17) and RS(19,7)+ZEM both show much shorter maximum tolerable refresh periods compared to Sparrow ECC. When the proportion of 1⟶0 bit errors are 100% and 99%, RS(19,17) shows maximum tolerable refresh periods of 220 ms and 180 ms, on average, respectively, while RS(19,17)+ZEM shows maximum tolerable refresh periods of 340 ms and 240 ms, on average, respectively. When the proportion of 1⟶0 bit errors decreases from 100% to 99%, both ECC schemes cannot tolerate the 1% 0⟶1 bit errors, resulting in shorter maximum tolerable refresh periods. In DNN weights, since most of the absolute values of weights are less than 1 (i.e., the exponent values are less than 127), they are already very small (i.e., close to 0) [21]. Thus, weights are much sensitive to 0⟶1 bit errors than 1⟶0 bit errors; 1⟶0 bit errors (even in the exponent) only decrease the already small values, while 0⟶1 bit errors, especially those in the exponent MSBs, dramatically increase the values by up to 2128X the original value, resulting in anomalously large weight value change. RS(19,17)+ZEM shows a slightly longer maximum tolerable refresh period compared to RS(19,17), since the encoding method of ZEM increases the number of '0' bits in the weights. With dominant 1⟶0 bit errors, RS(19,17)+ZEM encounters fewer errors than RS(19,17) due to the increased number of '0' bits. However, since the increased number of '0' bits leads to a higher probability of 0⟶1 bit errors in the exponent, RS(19,17)+ZEM cannot further extend the refresh period.

No protection also shows much shorter maximum tolerable refresh periods compared to Sparrow ECC. When the proportion of 1⟶0 bit errors are 100% and 99%, no protection shows maximum tolerable refresh periods of 860 ms and 32 ms, on average, respectively; the much shorter maximum tolerable refresh period of 99% 1⟶0 bit errors is due to the 0⟶1 bit errors in the exponent. When the 1⟶0 bit error proportion is 100%, no protection shows a longer maximum tolerable refresh period compared to RS(19,17) and RS(19,17)+ZEM, due to the micorrections of RS code [27]. In the RS code based schemes, though there are only 1⟶0 bit errors, when 'uncorrectable' errors



*The error bars denote the range of read latencies for each case.

Figure 7: Average memory read latency depending on ECC schemes across DNN models.

occur, the RS code may misdetermine the error as 'correctable', thus miscorrecting a data bit. In this case, when a '0' bit is miscorrected, a 0⟶1 bit flip occurs; according to our experiments, a 0⟶1 bit flip due to miscorrection occurs at refresh periods longer than 300 ms. Therefore, though there are only 1⟶0 bit errors, a 0⟶1 bit flip occurs in the exponent by miscorrection in RS code, thereby leading to significant accuracy loss.

Reflecting the maximum tolerable refresh period depending on ECC schemes shown in Figure 6(a), we evaluate the refresh energy depending on ECC schemes, normalized to no protection. As shown in Figure 6(b), when the proportion of 1⟶0 bit errors is 100%, Sparrow ECC reduces the refresh energy by 75.71%, 93.79%, and 90.40%, on average, compared to no protection, RS(19,17), and RS(19,17)+ZEM, respectively. The other ECC schemes consume higher refresh energy compared to no protection, since they have a shorter maximum tolerable refresh period. When the proportion of 1⟶0 bit errors is 99%, Sparrow ECC reduces the refresh energy by 99.10%, 94.92%, and 93.22%, on average, compared to no protection, RS(19,17), and RS(19,17)+ZEM, respectively. In summary, Sparrow ECC reduces refresh energy significantly by tolerating longer refresh period regardless of the proportion of error type, since it is more robust to 0⟶1 errors as well as 1⟶0 errors, compared to the other schemes.

5.3 Average Memory Read Latency

In DRAM-based memory systems, the memory read latency is determined by row activation latency (t_{RCD}), column activation latency (t_{CL}), and row precharge latency (t_{RP}) [6]. When adopting ECC, ECC decoding latency lies on the critical path of t_{CL}, thus leading to an increase in the average memory read latency. Moreover, the refresh operations also increase the average memory read latency by blocking read operations [6]. Reflecting the ECC decoding latency and maximum tolerable refresh period for each ECC scheme to our simulation environment, we analyze the average memory read latency across DNN models.

Figure 7 shows the average memory read latency of the ECC schemes across the DNN models. When the proportion of 1⟶0 bit errors is 100%, Sparrow ECC shows negligible overhead (1.21%, on average) in the average memory read latency, compared to no

Table 2. Area and power overhead of Sparrow ECC.

Component	Encoder	Decoder
Area	0.01 mm^2	0.01 mm^2
Power*	4.30 mW	4.46 mW

*The sum of dynamic power and leakage power.

protection, while consuming much lower refresh energy, due to the longer maximum tolerable refresh period. More importantly, Sparrow ECC reduces the average memory read latency by 13.12% and 11.23%, compared to RS(19,17) and RS(19,17)+ZEM, respectively, due to the 1) shorter decoding latency and 2) longer maximum tolerable refresh period. Sparrow ECC needs only 1 cycle for decoding latency, since it decodes only exponent delta values and parity bits; RS(19,17) and RS(19,17)+ZEM need 8 cycles for decoding latency, since RS(19,17) decodes full weight values. Thus, Sparrow ECC brings much less decoding latency than the other ECC schemes. Furthermore, since Sparrow ECC adopts much longer refresh period, it mitigates refresh-induced latency overhead, and thus further reduces average read latency compared to the other ECC schemes.

When the proportion of 1⟶0 bit errors is 99%, Sparrow ECC exhibits shorter average memory read latency than the other schemes, thanks to the much longer maximum tolerable refresh period. When the proportion of 1⟶0 bit errors is 99%, Sparrow ECC reduces the average memory read latency by 32.71%, 13.41%, and 11.55%, compared to no protection, RS(19,17), and RS(19,17)+ZEM, respectively. The results of latency reductions are in line with those of the maximum tolerable refresh period since the refresh operations block the normal read requests, negatively affecting the average memory read latency.

5.4 Area and Power Overhead

We evaluate the area and power overhead of Sparrow ECC with the SAED 14 nm FinFET [20] logic process technology. Instead of the 45 nm library used to evaluate the encoding and decoding latencies, we use the 14 nm library to consider the more advanced recent DRAM process [14]. According to our implementation results, as shown in Table 2, the total area and power consumption of Sparrow ECC are 0.02 mm^2 and 8.76 mW, respectively. Since the total area of one HBM3 stack is 121.00 mm^2 [24], Sparrow ECC has a negligible area overhead (0.02%). In case of power consumption, Sparrow ECC incurs negligible power overhead compared to the average power consumption of HBM2 (i.e., >20 W [13]).

6 CONCLUSION

In this paper, we propose Sparrow ECC, a lightweight ECC scheme for off-the-shelf HBM to reduce the refresh overhead while preserving DNN inference accuracy. Sparrow ECC mitigates retention errors due to the extended refresh period by limiting the exponent value range as well as adopting robust ECC to DNN weights. In each weight, Sparrow ECC corrects 1-bit error in the three replicas of the 1-bit sign and 1-bit error in the 4-bit of delta. In addition, in a weight block, Sparrow ECC corrects up to 3-bit errors in the seven replicas of each bit in the 8-bit maximum exponent and 1-bit flag, respectively, with no additional storage overhead compared to the off-the-shelf HBM3. When the 1⟶0 bit error proportion is 100% and 99%, Sparrow ECC extends average refresh period by 10X and 15X, reducing average refresh energy by 90.40% and 93.22%, respectively, compared to RS(19,17)+ZEM. Furthermore, thanks to the lower decoding latency and longer refresh period, when the 1⟶0 bit error proportion is 100% and 99%, Sparrow ECC reduces average memory read latency by 11.23% and 11.55%, respectively, compared to RS(19,17)+ZEM.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2020 R1A2C2003500 and No. RS-2023-00212711), Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00441-001, Memory-Centric Architecture Using the Reconfigurable PIM Devices), Samsung Electronics Co., Ltd (project IO201212-08130-01). Sung Woo Chung and Young-Ho Gong are the co-corresponding authors of this paper.

REFERENCES

[1] Michael Anderson, et al. 2021. First-Generation Inference Accelerator Deployment at Facebook. *arXiv:2107.04140.*

[2] Hyeong Kon Bae, et al. 2023. Twin ECC: A Data Duplication Based ECC for Strong DRAM Error Resilience. In *DATE.*

[3] Bita Darvish Rouhani, et al. 2020. Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point. In *NeurIPS.*

[4] Jacob Devlin, et al. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805.*

[5] Alexey Dosovitskiy, et al. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929.*

[6] Young-Ho Gong and Sung Woo Chung. 2016. Exploiting Refresh Effect of DRAM Read Operations: A Practical Approach to Low-Power Refresh. *IEEE Trans. Comput.* 65, 5.

[7] Kaiming He, et al. 2016. Deep Residual Learning for Image Recognition. In *CVPR.*

[8] Wei Huang, et al. 2024. BiLLM: Pushing the Limit of Post-Training Quantization for LLMs. *arXiv:2402.04291.*

[9] Myeongjae Jang, et al. 2022. ENCORE Compression: Exploiting Narrow-width Values for Quantized Deep Neural Networks. In *DATE.*

[10] Norman P. Jouppi, et al. 2021. Ten Lessons From Three Generations Shaped Google's TPUv4i: Industrial Product. In *ISCA.*

[11] Sehoon Kim, et al. 2023. Squeezellm: Dense-and-Sparse Quantization. *arXiv:2306.07629.*

[12] Taehwan Kim, et al. 2023. Thermal Improvement of HBM with Joint Thermal Resistance Reduction for Scaling 12 Stacks and Beyond. In *ECTC.*

[13] Taehwan Kim, et al. 2022. Thermal Modeling and Analysis of High Bandwidth Memory in 2.5 D Si-interposer Systems. In *iTherm.*

[14] N-H Lee, et al. 2022. Transistor Reliability Characterization for Advanced DRAM with HK+MG & EUV process technology. In *IRPS.*

[15] Young Seo Lee, et al. 2022. Stealth ECC: A Data-Width Aware Adaptive ECC Scheme for DRAM Error Resilience. In *DATE.*

[16] Shang Li, et al. 2020. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *IEEE Comp. Archit. Lett.* 19, 2.

[17] Yong Liu, et al. 2022. New Insight into the Aging Induced Retention Time Degraded of Advanced DRAM Technology. In *IRPS.*

[18] Chi-Keung Luk, et al. 2005. Pin: building customized program analysis tools with dynamic instrumentation. *ACM SIGPLAN Not.* 40, 6.

[19] Deepak M. Mathew, et al. 2018. An analysis on retention error behavior and power consumption of recent DDR4 DRAMs. In *DATE.*

[20] Vazgen Melikyan, et al. 2018. 14nm Educational Design Kit: Capabilities, Deployment and Future. In *SSSS.*

[21] Duy-Thanh Nguyen, et al. 2019. St-DRC: Stretchable DRAM Refresh Controller with No Parity-overhead Error Correction Scheme for Energy-efficient DNNs. In *DAC.*

[22] Duy-Thanh Nguyen, et al. 2021. ZEM: Zero-Cycle Bit-Masking Module for Deep Learning Refresh-Less DRAM. *IEEE Access* 9.

[23] Shailja Pandey, et al. 2023. NeuroCool: Dynamic Thermal Management of 3D DRAM for Deep Neural Networks through Customized Prefetching. *ACM Trans. Design Autom. Electr. Syst.* 29, 1.

[24] Myeong-Jae Park, et al. 2022. A 192-Gb 12-High 896-GB/s HBM3 DRAM With a TSV Auto-Calibration Scheme and Machine-Learning-Based Layout Optimization. *IEEE J. Solid-State Circuits* 58, 1.

[25] Adam Paszke, et al. 2019. Pytorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS.*

[26] Salvatore Pontarelli, et al. 2014. Low Delay Single Symbol Error Correction Codes Based on Reed Solomon Codes. *IEEE Trans. Comput.* 64, 5.

[27] I. Sofair. 2000. Probability of miscorrection for Reed-Solomon codes. In *ITCC.*

[28] James E. Stine, et al. 2007. FreePDK: An Open-Source Variation-Aware Design Kit. In *MSE.*

[29] Ashish Vaswani, et al. 2017. Attention is All you Need. In *NeurIPS.*

[30] Yonghui Wu, et al. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv:1609.08144.*

[31] Yang Zhao, et al. 2019. Memory Trojan Attack on Neural Network Accelerators. In *DATE.*

[32] JEDEC. 2023. JESD238A, High Bandwidth Memory DRAM (HBM3). Retrieved from https://www.jedec.org/standards-documents/docs/jesd238a.

[33] Nvidia. 2024. NVIDIA H100 Tensor Core GPU. Retrieved from https://www.nvidia.com/en-us/data-center/h100/.