Perturbation-Based Error Detection and Correction (PBEDC) in Dependable Large-Scale Machine Learning Systems

Ziheng Wang, Pedro Reviriego, Shanshan Liu, Farzad Niknia, Xiaochen Tang, Zhen Gao, and Fabrizio Lombardi

***Abstract*—Conventional error-tolerant schemes for Neural Networks (NNs) usually require either redundancy, or changes in normal operation, leading to considerable overheads. They are not feasible for large-scale Machine Learning (ML) systems that typically employ several complex networks. This paper proposes a Perturbation-Based Error Detection and Correction (PBEDC) scheme designed to perform error detection and correction by reutilizing the inference process. Dependable performance is defined by the ability to operate correctly in the presence of errors and is a key characteristic under consideration. PBEDC employs a compact set of representative samples that are selected to monitor a few check nodes with intermediate signals. The effectiveness of PBEDC is evaluated by taking Contrastive Language-Image Pre-Training (CLIP) networks as a case study. Compared with traditional schemes that use the final prediction as the check node, PBEDC achieves a superior error detection rate (> 95%) and can handle single bit-flip errors in the weights (which cannot be captured in existing schemes). This also enables the correction of errors when the proposed scheme is combined with the use of parity codes. Furthermore, in this paper, the analysis and simulation results show that the number of PBEDC samples required for achieving a satisfactory error tolerance is very small; the complexity of the proposed scheme does not scale up with the network size and this advantage is very pronounced with large-scale ML systems.**

***Keywords*—Error Detection, Error Correction, Large-Scale Neural Networks, Soft Errors, CLIP.**

I. Introduction

The complexity of neural network (NN) systems has significantly grown in the last few years, leading to implementations that have billions of parameters. NNs have been employed in a diverse array of fields, notably in domains such as computer vision [1], natural language processing [2], image-text matching [3]. Given the intricate network structures and voluminous data involved, the detection and correction of hardware errors that corrupt the model are critical to guarantee accurate operations of these large-scale machine learning (ML) systems. Dependability is the feature by which a system continues to provide correct results also in the presence of errors; schemes have been studied to achieve error detection or correction such as utilizing redundancy of computational neurons [4], [5] and error correction codes (ECCs) [6]. However, with the increasing scale of ML systems, the reliance on traditional error detection mechanisms, such as those incorporating hardware redundancy, induces considerable overhead. Therefore, the use of protection techniques, where the ML system itself is used for error detection, emerges as a compelling solution. This approach enables dependable operation while simultaneously mitigating hardware overhead.

An example of a large ML model is Contrastive Language-Image Pre-Training (CLIP), an NN capable of learning visual concepts from natural language inputs [7]. CLIP establishes relationships between texts and images and has been widely used in text-to-image tools such as DALL-E [9]; moreover, it can be also applied to zero-shot learning [7]. The architecture of CLIP incorporates two branches or subnetworks - an image encoder and a text encoder. The final output measures the similarity between the images and texts to generate a loss or make predictions. The trained model is typically preserved in memory as weight matrices. Thus, the detection or correction of hardware errors during inference, especially for those in the memory corrupting the weights, is critical for dependable operation and preventing corrupted results. Given CLIP's characteristics as a large-scale model utilizing deep NNs and vast volumes of data, a cost-effective error detection (or correction) technique becomes essential to assure its dependability.

Low-cost error detection solutions such as so-called self-test schemes have been proposed for NNs [8]. Such techniques utilize a selected set of samples for which the inference results have been precomputed and stored; therefore, errors during inference can be detected by comparing the results with those stored ones [8], [10]. Since a self-test technique reuses existing hardware, it is more attractive for protecting large-scale models such as CLIP when compared to traditional redundancy-based schemes. However, relying on a change in inference results can only detect errors with certain inputs (for which the pre-computed results are stored). To achieve high error detection

Manuscript received August 14, 2023 and revised January 10, 2025. The research was supported by the Spanish Agencia Estatal de Investigación under Grant PID2019-104207RB-I00 and Grant TSI-063000-2021-127, and the NSF under Grant CCF-1953961 and 1812467. *Corresponding author: Shanshan Liu* (email: ssliu@uestc.edu.cn).

Ziheng Wang, Farzad Niknia and Fabrizio Lombardi are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, USA.

Pedro Reviriego is with the Departamento de Ingeniería de Sistemas Telemáticos, Escuela Técnica Superior de Ingeniería de Telecomunicación, Universidad Politécnica de Madrid, Madrid 28040, Spain.

Shanshan Liu and Xiaochen Tang are with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, 611731, China.

Zhen Gao is with the School of Electrical and Information Engineering, Tianjin University, Tianjin 300072, China.

coverage, the execution of considerable input samples must be required in the self-test scheme. Furthermore, such a scheme could potentially leave some errors undetected, as an error may not have a sufficiently influential impact to alter the inference result for one sample but might change the result for another.

The above features of the existing self-test technique motivate this paper to investigate a more efficient approach that can keep the low-cost benefits of self-test but improve its error detection performance. A crucial insight gleaned from the self-test technique is that for effective error detection, monitoring a change in the classification result is not mandatory. Instead, a so-called perturbation (i.e., a corrupted value that is different from the error-free case) at any node in the model implies in most cases the presence of an error. Based on this observation, a Perturbation-Based Error Detection and Correction (PBEDC) scheme is proposed in this paper to efficiently protect large-scale ML models against errors. Different from existing self-test schemes introduced previously, in PBEDC, a set of internal nodes are selected for monitoring the model, and their error-free values for a few input samples are stored. Then, error detection is performed by running inference on these input samples and checking whether a perturbation occurs at any of the nodes. In particular, the existing self-test technique of [8] can be considered as a special case of PBEDC, because it only monitors the output nodes (i.e., the inference results).

The proposed scheme can also be extended to perform single-bit error correction for the ML model parameters when used in conjunction with parity codes. In such a scenario, when a parity code detects an error, the check samples can be used to pinpoint the erroneous bit. If the error induces a perturbation at the check nodes for some of the test samples, flipping a bit in the erroneous parameter and rechecking the samples enables the verification of a perturbation presence and the subsequent correction of the error.

To assess the advantages of the proposed PBEDC scheme when protecting large-scale ML models, we consider the CLIP networks used for zero-shot classification tasks as a case study and most frequently occurred single-bit memory errors as the error model. The simulation results demonstrate that PBEDC exhibits superior performance in terms of error detection and correction. The major contributions of this paper are summarized as follows:

1) A new method, namely PBEDC, is proposed for error protection in large-scale ML systems at a low cost. For error detection, PBEDC monitors only the selected internal nodes and applies a small set of representative samples (referred to as PBEDC samples) to detect perturbations caused by errors; this scheme relies on the use of the original network, so avoiding the use for additional redundancy. PBEDC is also capable of correcting single-bit errors when the model parameters are protected with a parity code.
2) The performance of PBEDC is evaluated on a CLIP model by error-injection simulations and compared with the existing self-test method of [8]. The results show that PBEDC has a significantly higher coverage for single-bit error detection (> 95%) with only 10 PBEDC samples when applying softmax vectors as check nodes. Moreover, by using a parity bit to protect the model parameters, all detected errors can be located and corrected by the extended PBEDC error correction scheme.
3) The number of PBEDC samples and the size of the so-called synthetic detection matrix (used to select PBEDC samples) required for error protection are discussed. The analysis and simulations indicate that appreciable error tolerance is achieved with a minimal number of samples; this advantage becomes evident when considering large-scale evaluation sets.

The rest of the paper is organized as follows. Section II provides some preliminaries. In section III, the proposed PBEDC technique for both error detection and correction is presented; its implementation details and performance evaluation are presented in section IV. The paper concludes with Section V, which encapsulates the main findings and avenues for future research.

II. Preliminaries

This section provides a concise introduction to large-scale ML systems, focusing specifically on the CLIP model. The underlying error model presumed in this study is also detailed. Furthermore, existing error detection/correction schemes that are closely related to the proposed scheme, are also reviewed.

A. Large-scale ML systems

The overview of the trends in large-scale ML systems has been extensively discussed in [11], with particular emphasis on prominent applications such as DALL-E [9] or ChatGPT [12]. Their operation usually utilizes diverse deep NNs and multiple encoders/decoders, contributing to their high complexity. Error tolerance design is a challenging problem for these large-scale applications, especially on resource-limited platforms; this occurs because powerful protection techniques often introduce redundancy that further increases the hardware overhead, while low-cost schemes cannot offer a satisfactory protection performance.

As an example of a large-scale ML system, CLIP is considered in this paper. CLIP is composed of text and image encoders; during training, texts and images are input into these two encoders. For the image encoder, a modified Resnet (a type of cascade convolutional NN with skip connections) is used; it is similar to the original version of Resnet [13] with some modifications as described in [7]. The text encoder uses a transformer operating on a lower-case byte pair to encode the representation of the text as described in [14]. The outputs of the images/text encoders are converted to an identical format and the cosine contrastive loss is computed[1]. The model is trained to minimize the cosine distance between the given images and their corresponding labels. CLIP models typically use deep NNs (with more than 50 layers) and extremely large

[1] For further mathematic details, please refer to [7] and [15].

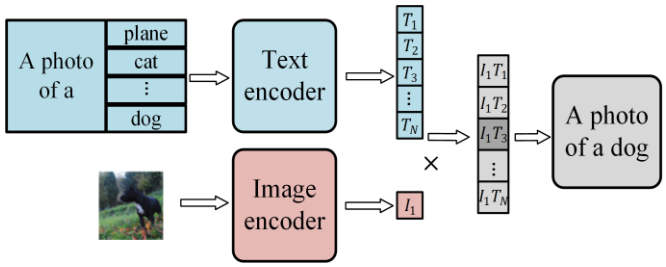


Fig. 1. The inference process of the CLIP network in zero-shot prediction.

datasets; therefore, training requires a large computational effort.

After establishing the model (i.e., image and text encoders), it can be used for inference as shown in Fig. 1. The notation N is used to represent the number of classes. For the classification of zero-shot learning, the labels are treated as descriptive texts (such as "a photo of [label]") and they are encoded by the trained CLIP model. The same cosine distance is calculated for the inference image and all candidate labels; then, the prediction is made by the pair with the smallest distance.

B. Error model

During inference, the trained models are stored in a memory as weight (and bias) matrices and the underlying hardware is prone to suffer from different types of errors; one of the most common types is the so-called soft error, as mainly caused by radiation particles [27]. Errors can flip the stored bits and then cause data corruption; thus, they are considered as a significant reliability/dependability issue for nanometric memories. If corrupted values are utilized during the inference of an input sample, it could lead to an incorrect prediction, potentially causing system failure. In large-scale ML systems (such as the CLIP models) due to the complexity of the NNs, small errors can propagate and spread to deeper stages and thus, they may cause significant degradation in performance; therefore, error detection and correction during inference is of significant interest for dependable operation.

The error model considered in this paper is the most frequently occurring single bit-flip error in the weight matrices; this phenomenon models the effect of soft errors on memories that store the parameters. It is realistic because when considering the low error rate in practice, single-bit soft errors can cover most cases. Additionally, other scenarios (such as multiple bit-flips in one memory word or multiple words with a bit-flip) should be easier to detect by the proposed scheme because they in general introduce more perturbations; hence, the single-bit error should be the most challenging error detection case.

The parameters of the CLIP model are represented in the IEEE 754 standard half-precision floating-point format [16] (Fig. 2) and stored in a memory with 16-bit words. The decimal value of a half-precision floating-point number is calculated as

$$Value_{dec} = (-1)^S \times 2^{E-15} \times (1 + M \times 2^{-10}) \qquad (1)$$

where S, E, and M are the decimal number represented by the corresponding sign, exponent, and mantissa bits. According to (1), a bit-flip in the sign or exponent bits leads to more significant changes in value.

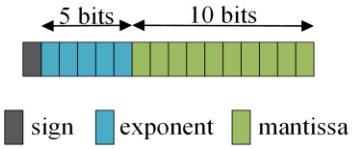


Fig. 2. Data represented in IEEE standard 754 half-precision floating-point format.

C. Related work

The error tolerance of NNs has been widely studied, and popular dependability schemes can be classified into the following types: *1)* Adding redundancy, which includes duplicating the critical neurons/layers [4], [5], [17]-[19] or introducing extra bits on the parameters for ECCs [6], [20]; *2)* Changing the ML algorithm, which includes adding penalty terms and modifying the weights [21]-[24]; *3)* Retraining the model with artificial errors [19], [25], [26].

Among these schemes, redundancy-based techniques can lead to considerable overhead; for example, one (five) ECC bits are required for each 16-bit half-precision floating-point data to achieve single-bit error detection (correction). Moreover, the ECC decoding process must be performed prior to using each parameter, which significantly affects the computational efficiency of the NNs. Consider the CLIP network that has over 23 million parameters; more efficient schemes that incur lower costs are more attractive. For the second type of scheme, a change in the NN, either in data representation or algorithm, is not only hard to implement but may also lead to unpredictable degradation. Moreover, the third type of scheme that retrains robust NNs is rather impracticable considering the large-scale data/network. Also, all these schemes have only been verified on very small networks (mostly on a simple 3-layer Multilayer Perceptron); therefore, reusing the NN for error protection (without changing their normal operational scheme) would be a more efficient choice for dependable large-scale ML systems.

The closest technique to the proposed scheme is the self-test of [8]. Self-test uses a set of input samples to detect errors by re-running inference and comparing with the pre-stored results. Since it considers only the prediction changes of the ML system to detect errors, the self-test has limited ability to detect some errors; for example, in [8] a bit error rate of at least 10^{-4} (which for a large model implies that thousands of bit-flips occur) is considered to capture their impact on the results. Conceptually, such a self-test scheme can be seen as a particular case of the proposed PBEDC technique in which the nodes monitored for perturbation are the outputs of the ML model. In PBEDC, any nodes in the model can be selected to detect perturbations as discussed in the next section to show that it results in significantly better error detection/correction capability.

III. Perturbation-based Error Detection and Correction (PBEDC)

The proposed PBEDC scheme is based on pre-computing and storing the values of a set of nodes during inference for a

group of input samples (i.e., PBEDC samples). Subsequently, the model undergoes an inference run and the node values are compared to ascertain the presence of any perturbations, indicative of an error occurrence. This scheme relies on error propagation to the check nodes for error detection; so, this section first discusses error propagation in NNs. The selection of check nodes in PBEDC is then illustrated and its difference from the existing self-test scheme of [8] is discussed. The detailed scheme is presented next; it shows the selection of PBEDC samples and their application to the evaluation sets. Then, the use of PBEDC for error correction is discussed when combined with a parity code. Finally, the complexity of implementing the main steps in both the error detection and correction schemes is analyzed.

A. *Error propagation*

Error detection is conditioned on the error impacting the value of at least one check node; this happens as the error propagates through the system's operations to a check node. For analyzing propagation, we can take as an example a single neuron computation in an NN as shown in Fig. 3, where x_i denotes the inputs and w_{ij} denotes the weights (the bias b can be included in the weights as an extra dimension of input 1); r is the dimension of the current layer and j is the node index in the next layer. Consider an error e in a product of $w_{ij}x_i$ that changes it to $(w_{ij} + e)x_i$, then the multiply-accumulation result $y_j(x)$ is modified to $y_j(x) + e \cdot x_i$, i.e., with an additive error term; it will propagate through the activation function Φ and the following layers, such that the error will spread and affect more neurons.

Intuitively, errors can propagate to the outputs and make changes to the network; however, several conditions may stop error propagation, as analyzed in more detail next.

- *Numerical resolution*: when the error is smaller than the resolution used for the internal signals, it may be removed as part of the truncation/rounding process performed in the arithmetic operations.
- *Saturation*: the value may saturate, for example, when using a ReLU activation function and the input value is negative, then the output value is zero regardless of the magnitude of the input. The same applies to arithmetic operations that are already saturated in the error-free case.
- *Zero input value:* In this case, an error in the weight is not propagated (multiplication by zero).
- *Error compensation:* As errors propagate to different layers, then they may eventually compensate, so returning to the original values.

Even though the above cases have the potential to prevent error propagation, they typically occur with very low probability in most applications, thus leading to an expectation of extensive error propagation. As demonstrated by the simulation results provided in Section IV.B, the occurrence of an error that leaves the softmax vector unchanged over the entire evaluation set is exceptionally rare. In addition, errors occurred in these scenarios are not harmful because they do not change the final classification results, so they do not need to be handled.

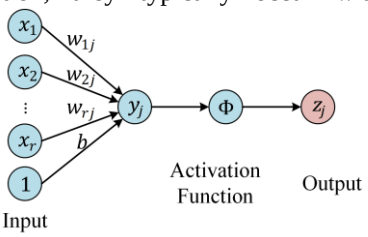


Fig. 3. Forward propagation of a single fully-connected neuron in an NN.

In this paper, error injection simulates single-bit errors by randomly selecting a weight in the neural network and a bit position within its binary representation to flip. The modified weight is then reintegrated into the model, enabling the evaluation of PBEDC's ability to detect and correct errors under realistic and random fault conditions.

B. *Check nodes selection*

Based on the error propagation process previously described, the selection of the check nodes close to the outputs in general would be a good strategy, because errors propagate towards the outputs; the use of the nodes just before the outputs as check nodes could in principle detect all errors that propagate in the network. This paper chooses the softmax values prior to the final prediction as check nodes; this combines the propagation of both the image and the text encoders, and its small size leads to reduced overhead. Fig. 4 illustrates the choice of check nodes that are used hereafter in this paper and compares them with the existing self-test scheme [8]. In particular, if a rough detection/correction is preferred, the proposed scheme can be also applied to the output check node for only monitoring errors changing the predictions (i.e., similar to the self-test scheme).

C. *PBEDC scheme: error detection*

In the proposed scheme, a small set of PBEDC samples (with a cardinality of k) is utilized. The intermediate outputs of the error-free model with these samples are utilized for error detection. The entire procedure consists of the following steps:

***Step 1*:** Run the error-free model with m randomly selected samples; save the softmax outputs of all N classes after the image and text encoders for each sample as "golden" results.

***Step 2*:** Inject single-bit errors and rerun the CLIP model with the same set of samples as that in step 1; note that each run incorporates only one injected error within the model parameters. Compare the softmax outputs of every sample with the corresponding golden values. If there are any changes in the

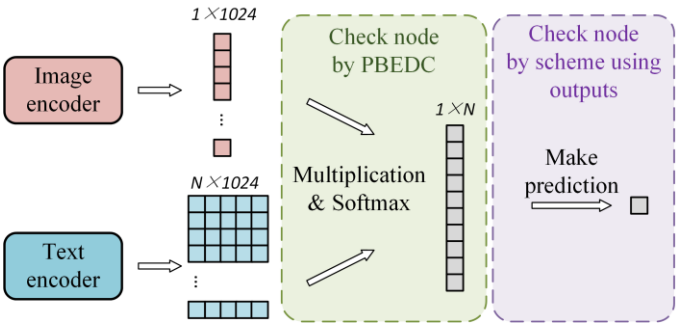


Fig. 4. Comparison for check nodes in the CLIP network used by the proposed PBEDC scheme and the self-test scheme of [8] (N denotes the number of classes).

outputs, store a 1 in the corresponding position of a vector of size m; otherwise, store a 0.

***Step 3*:** Repeat step 2 for n trials with different injected errors. This generates a matrix to denote whether the output is changed; this matrix of size $m \times n$ is called the synthetic detection matrix. Note that this matrix is only used to select PBEDC samples prior to inference and thus, it does not introduce any computational overhead; the size of the matrix is analyzed in Section IV.

***Step 4*:** Determine the set of PBEDC samples that can cover most (preferably all) errors. This can be accomplished by using the following two heuristic strategies.

(i) ***Greedy search*:** A greedy search algorithm sequentially selects k PBEDC samples, and the selection of each sample is conditioned to maximize the coverage of columns containing a "1" in the corresponding synthetic detection matrix. Columns that have already been covered in previous steps are disregarded in the ensuing search iterations. The greedy search strategy is implemented to ensure that the selected k samples encapsulate, to the greatest extent possible, the diversity of error-induced changes, as represented by more unique columns containing a "1". Although it might not consistently yield the optimal solution, the adoption of the greedy search is warranted by its computational efficiency in large-scale systems.

(ii) ***Max overlapped coverage:*** In this case, the samples are sorted in descending order based on the number of columns containing a "1" in the synthetic detection matrix. The top-k samples are then selected, ensuring that the chosen samples provide maximal overlapped coverage of the columns containing a "1". This strategy ensures that each selected sample is sufficiently representative of the error-induced changes. Despite the total count of covered errors potentially being lower than that of a greedy search due to overlap, the robust coverage of each individual sample on errors is preserved.

The above-described steps are also illustrated in Fig. 5. The pink cells in the matrix refer to values that are different from the error-free ones for this specific test sample and error; with at least one such cell in a row, the corresponding sample is able to detect the error. Then for each sample and error, a column is added to the synthetic detection matrix (in red samples that detect this error). Finally, k samples are selected by the proposed PBEDC algorithm. In this illustration, we assume that $k = 3$ PBEDC samples are selected from the $m \times n$ detection matrix. Specifically, samples "e", "c", and "d" are selected in order by the greedy search strategy ("a" is not selected because it fully overlaps with "e"); samples "e", "b", and "a" are selected by the max overlapped coverage strategy.

Once the PBEDC samples have been selected, they can be used for error detection by observing alterations in the check nodes upon these samples. Depending on the requirement of error tolerance of a specific ML system, the PBEDC samples can be periodically inserted into the inference dataset for real-time error detection at a given frequency. For instance, as evaluated in section IV, inserting 10 PBEDC samples for every 10,000 inference samples appears to be a feasible strategy, resulting in an overhead of about 0.1%. In this case, the additional computational overhead introduced by the proposed scheme is limited solely to the inference of the 10 PBEDC samples, which is substantially low. To better introduce the above steps of the proposed PBEDC scheme, the pseudocode is presented as Algorithm 1.

Algorithm 1: Error detection using PBEDC scheme

Input: Error-free CLIP model; m randomly selected input samples; the number of error injection trials n, the size of PBEDC samples k.
Output: Boolean variable *Error_Flag* indicating if errors are detected in the network.

{***Initialization***}
Initialize *golden_values*[i] = ∅ **for** i = 1 **to** m
Initialize *detection_matrix*[i][j] = 0 **for** i = 1 **to** m **and** j = 1 **to** n
Initialize *PBEDC_samples* = ∅
Initialize *Error_Flag* = **False**
{***Generate golden values***}
for i = 1 **to** m **do**
    golden_values[i] ← **Inference**(error-free model, sample i)
end for
{***Create detection matrix***}
for j = 1 **to** n **do**
    Inject single-bit error into model
    for i = 1 **to** m **do**
        detection_matrix[i][j] ← (**Inference**(erroneous model, sample i) ≠ *golden_values*[i])
    end for
end for
{***Select PBEDC samples***}
if using Greedy Search **then**
    While size(*PBEDC_samples*) < k **do**
        Choose sample maximizing covered errors in *detection_matrix* and add to *PBEDC_samples*
    end while
else if using Max Overlapped Coverage **then**
    Sort samples by number of "1"s in *detection_matrix* rows
    Select top-k samples and add to *PBEDC_samples*
end if
{***Real-Time Error Detection***}
Insert *PBEDC_samples* into inference dataset periodically
for each *test_sample* in *PBEDC_samples* **do**
    if Inference(model, *test_sample*) ≠ *golden_values*[*test_sample*] **then**
        Error_Flag = **True**
    end if
end for
Return *Error_Flag*

If necessary, extra approaches can be applied to protect the recorded check node values of the PBEDC samples, such as triple modular redundancy (TMR) or ECCs; given the extremely small number of PBEDC samples, the associated costs of these strategies are negligible.

Once errors are detected in the proposed scheme, correction approaches can be applied, such as reloading parameters if there

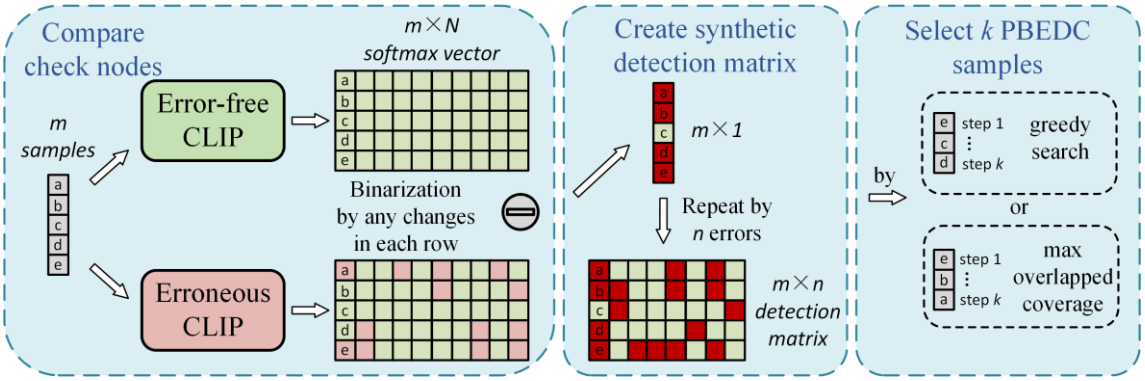


Fig. 5. The proposed PBEDC scheme for error detection.

is a backup. Alternatively, as will be elaborated in the subsequent subsection, the PBEDC samples can also be used to achieve error correction when the model parameters are protected by a parity code.

D. PBEDC scheme: error correction

The previously described scheme can only detect errors because the erroneous weight cannot be located. To extend it for error correction, the use of parity codes of the model parameters must be combined. Specifically, assume that the weights are stored with a parity bit that is checked when they are accessed during inference. Considering that introducing an extra parity bit can be costly in large-scale systems, it can be realized by substituting the least significant bit in the mantissa; this approach has been reported to cause negligible accuracy degradation [28].

As shown in Fig. 6, when an error is detected by the parity code, the PBEDC samples can be run to identify those on which there is a perturbation on the check nodes. If no perturbation is observed, the error must be trivial and it cannot be corrected; else, correction can be implemented by flipping a bit of the erroneous weight and rechecking the samples. If the perturbation disappears, it can be inferred that the flipped bit was the erroneous one, and thus, it is corrected. The bit-flipping correction can start from the most significant bits (MSBs, from sign, exponent, to mantissa), because errors on such bits are more likely to cause perturbations. If all 16 bits of a half-precision floating-point weight have been tried but the perturbance is still present, the process ends, which indicates that the error cannot be corrected (e.g., more than one erroneous bit in the weight).

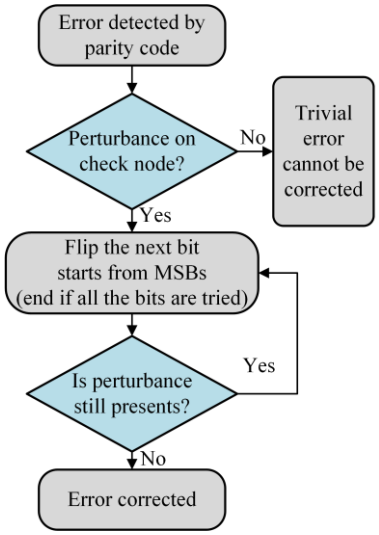


Fig. 6. Flow chart of the proposed PBEDC scheme for error correction.

The overhead for this error correction scheme is very low, because only one of the PBEDC samples with perturbation is required for locating the error and no additional memory redundancy is required for the parity code. Considering the high detection rate of the proposed error detection scheme (which will be shown in Section IV), the error detected by the parity code is very likely to be covered and thus, it can also be corrected. A mis-correction case may occur but intuitively, the probability of removing the perturbation by flipping a bit that is different from the one in error seems to be negligible. These expectations are confirmed by the simulation results presented in Section IV.

E. Complexity analysis: error detection

This subsection analyzes the complexity of the main steps in the proposed PBEDC scheme. The complexity of error detection with the PBEDC scheme in Algorithm 1 is summarized in Table I. The notation in Table I is as follows: m is the number of randomly selected samples; n is the number of error injection trials; k is the size of the PBEDC samples (usually significantly smaller than m and n); T_{inf} is the time required for inference of a single test sample; M is the output size of the check node.

TABLE I
COMPLEXITY ANALYSIS OF THE MAIN STEPS IN THE ERROR DETECTION SCHEME WITH PBEDC

Steps	Time complexity	Space complexity
Initialization	$O(m \cdot n)$	$O(m \cdot (N+n))$
Generate golden values	$O(m \cdot T_{inf})$	-
Create detection matrix	$O(m \cdot n \cdot T_{inf})$	-
Select PBEDC samples (Greedy Search)	$O(k \cdot m \cdot n)$	$O(k)$
Select PBEDC samples (Max overlapped coverage)	$O(m \cdot \log m + k)$	$O(m)$
Real-Time Error Detection	$O(k \cdot T_{inf})$	$O(k \cdot M)$
Overall (based on dominant terms)	$O(m \cdot n \cdot T_{inf})$	$O(m \cdot (N+n))$

As per Table I, for the greedy search strategy, the main time complexity terms are the construction of the detection matrix $O(m \cdot n \cdot T_{inf})$ and sample selection $O(k \cdot m \cdot n)$; for the max overlapped coverage strategy, the primary complexity terms only include $O(m \cdot n \cdot T_{inf})$, because sample selection is significantly lower in complexity. Generally, the time complexity of sample selection only depends on *k*, *m* and *n*, making it scalable for large models. Also, once the samples are prepared, the real-time error detection complexity is very low, mainly dependent on $O(k \cdot T_{inf})$. Compared with other self-test schemes [8] that require 30 test samples, the proposed PBEDC achieves a higher detection rate at a smaller number of *k* test samples (less than 10 samples, as validated in the evaluation section), proving the advantage in terms of computational effort.

The dominant factors in space complexity are the detection matrix $O(m \cdot n)$ and the storage of golden values $O(m \cdot M)$. For large-scale models and datasets, a reduction of the number of samples *m* and error injection trials *n* or the use of compression techniques can efficiently meet the storage requirements. However, as these numbers are significantly smaller than the total number of parameters in the entire network, the storage overhead is negligible.

Note that PBEDC is designed for large-scale ML systems in which model retraining is infrequent due to the high computational overhead. In this case, model updates are often limited to fine-tuning of parameters; thus, we suggest incrementally updating the golden values for only the modified parts of the model. This approach reduces the computational overhead while maintaining detection accuracy. Since this paper focuses on the basic scheme of PBEDC, the extended online update approaches will be investigated in future works.

Through this analysis, it is evident that PBEDC error detection has excellent scalability and low runtime overhead for application to large-scale neural networks.

F. Complexity analysis: error correction

As described in Section III.D, the PBEDC scheme achieves error correction by combining it with a parity code. This subsection analyzes the complexity of the main steps in the error correction case. Note that the steps listed in Table I (except for the error detection step) are still required to select the PBEDC samples, but differently, one of the PBEDC samples with perturbation is required for locating the error once it is detected by the parity code. The complexity of the additional steps for error correction with the PBEDC scheme is presented in Table II. The notations in Table II are as follows: *P* is the total number of parameters in the network protected by the parity code; *b* is the bit-width of the applied floating-point format.

TABLE II
COMPLEXITY ANALYSIS OF THE MAIN STEPS (IN ADDITION TO THE ERROR DETECTION SCHEME) IN THE ERROR CORRECTION SCHEME WITH PBEDC

Steps	Time complexity	Space complexity
Parity checking	$O(P)$	$O(P)$
Generate golden values	$O(b \cdot T_{inf})$	$O(N+b)$
Overall	$O(P + b \cdot T_{inf})$	$O(P+N+b)$

The total time complexity of the error correction part is $O(P + b \cdot T_{inf})$, and the space complexity is $O(P+N+b)$. Note that the complexity analysis is considering the worst case; since the bit-flipping correction can start from the MSBs (the more likely to cause perturbations by errors), the required trials should be smaller than the bit-width *b*. Moreover, considering the scenario protecting the entire large-scale network, the number of parameters *P* is usually very large, so the complexity term $O(P)$ dominates. In this case, the PBEDC scheme locates and corrects the error with minor additional overhead compared with the traditional parity scheme. If only the critical modules in the network are protected, hence the total overhead can be greatly reduced. This feature makes PBEDC more suitable for deployment in large-scale neural networks, especially in real-time systems or resource-constrained environments.

IV. SIMULATION RESULTS

A. Implementation details

In this paper, two popular image datasets CIFAR10 [29] and Mini-imagenet [30] are used during inference. CIFAR10 (with size 32×32 RGB images) has 10 different classes ($N = 10$) and Mini-imagenet (RGB images of several sizes) has 100 different classes ($N = 100$). Zero-shot classification by CLIP networks is conducted for both datasets and the golden results are recorded. The code that implements the proposed PBEDC scheme as well as the scripts to reproduce the experiments, is available in a public repository[2].

The CLIP network provides multiple pre-trained models, and this paper chooses the simplest, i.e., "RN50" (modified 50-layer Resnet) [7]. The output of CLIP includes the image encoder output and the text encoder output; the matrix $output_{image} \times output_{text}$ leads to a vector of size 10 (100) for CIFAR10 (Mini-imagenet). For each dataset, 10000 samples are designated as the test set for one PBEDC check phase. The softmax function is applied to the vector for classification. As it generates the output vectors wherein each element represents the probability of the corresponding class, the label with the largest value is selected as the final prediction.

As discussed in Section II.B, single bit-flip errors are chosen as the error model. Each bit-flip is injected into a random layer in the CLIP network with equal probability. The model uses a half-precision floating-point format, and the bit-flip errors are randomly injected to one of the 16 bits each time. In most cases, the network is not significantly affected (as errors in the non-significant bits), but occasionally it completely fails (due to errors in the significant bits) [31].

B. Error detection performance

To evaluate the error detection performance of the proposed PBEDC scheme, a synthetic detection matrix of size $m \times n$ is generated first for selecting the PBEDC samples. Specifically, for each of m randomly selected samples, n random bit-flip errors are injected into the CLIP network (only one error at a time); then, compared with the golden results, a $m \times n$ matrix is obtained indicating the changed outputs. The matrix size for

[2] https://github.com/ZihengWang-2/PBEDC

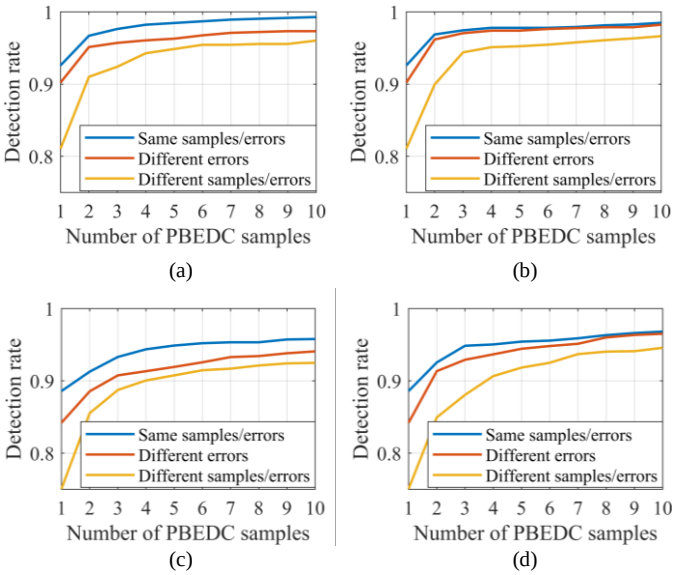


Fig. 7. Error detection rate of the proposed scheme using different number of PBEDC samples: (a) dataset CIFAR10 with *greedy search*; (b) dataset CIFAR10 with *max overlapped coverage*; (c) dataset Mini-imagenet with *greedy search*; (d) dataset Mini-imagenet with *max overlapped coverage*.

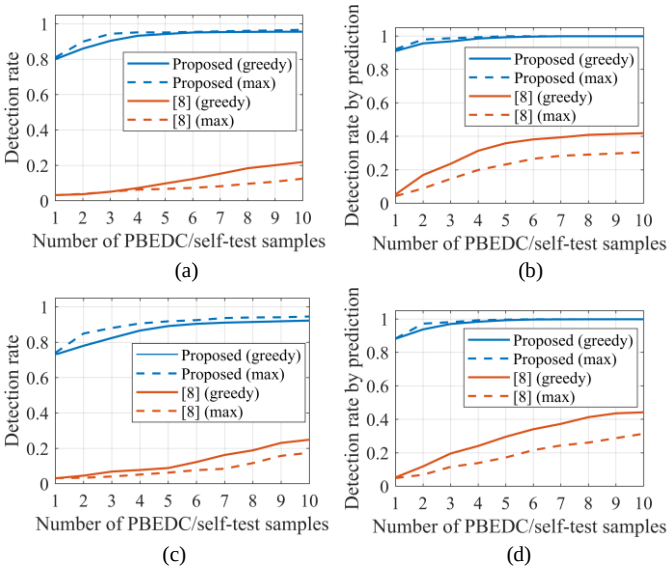


Fig. 8. Error detection performance of different schemes: (a) detection rate for dataset CIFAR10; (b) detection rate by prediction changes for dataset CIFAR10; (c) detection rate for dataset Mini-imagenet; (d) detection rate by prediction changes for dataset Mini-imagenet.

datasets CIFAR10 and Mini-imagenet is 1200×1000 and 1500×1000, respectively; this is suggested by the following subsections. The PBEDC samples are selected based on the synthetic detection matrix by using two strategies (as described in Section III.C): i) *greedy search* and ii) *max overlapped coverage*. To perform an extensive evaluation, simulation by error injection is conducted by considering three different evaluation sets (each has a size of $m \times n$):

- *Same samples/errors*: The evaluation set is the same as the synthetic detection matrix, i.e., using the same samples and injected errors.
- *Different errors*: The evaluation set uses the same samples as the synthetic detection matrix, but different errors are injected.
- *Different samples/errors*: The evaluation set uses a new group of m samples that are different from the synthetic detection matrix, and different errors are injected. This is considered as the general evaluation set.

In each evaluation set in the simulation, identical PBEDC samples are employed. The changes in the check nodes when applying these PBEDC samples signify the occurrence of an error within the evaluation set. The effectiveness of this approach is evaluated by the detection rate, defined as the percentage of individual single-bit errors identifiable by these samples. Moreover, the influence of the number of PBEDC samples on the error detection performance is assessed by varying k from 1 to 10. This can be accomplished by applying the top k samples as subsets within the chosen pool of 10 samples. The error detection rate results for both CIFAR10 and Mini-imagenet are shown in Fig. 7; for the evaluation sets with different errors or different samples/errors, the results are averaged for 100 runs.

As per Fig. 7, when comparing the three evaluation sets, the error detection rates of the samples decline with new samples/errors; however, a small number of PBEDC samples can still cover most of the errors. Consider the general scenario (different samples/errors) as an example; only 5 PBEDC samples cover more than 93% of the errors while 10 PBEDC samples cover more than 95% of the errors. Moreover, the *max overlapped coverage* strategy is shown to achieve a better performance than the *greedy search* strategy in the general case, so it provides a better generalization for error detection. We have also evaluated PBEDC samples that were randomly selected (without following either strategy); the detection rate performed significantly lower (<60%), which was not recommended.

To show the importance of monitoring internal check nodes, a second experiment is conducted to assess the error detection performance in the following two scenarios:

- *Detection rate*: the rate of detected errors among all injected errors.
- *Detection rate by prediction changes*: the rate of detected errors among only errors that lead to any changes in the final prediction.

In each case, the error detection performance of both PBEDC and the self-test of [8] is evaluated and compared. The general evaluation set with different samples/errors is considered in this experiment and the two strategies of selecting representative samples are employed.

As per the results given in Fig. 8, the proposed PBEDC detects most errors that change the predictions with only 3 or 4 samples, while the self-test of [8] provides a significantly lower error detection rate even if using more samples. Moreover, PBEDC can cover over 95% of the errors that only cause a perturbation in the interval nodes (softmax vectors); this number reduces to lower than 20% if the synthetic detection matrix is generated by the changed predictions as in [8]. This is reasonable considering that the matrix generated based on the changed predictions is very sparse (only 4.84% of 1s), therefore it is difficult to select representative samples. In addition, the

proposed scheme is very reliable; according to our experiments, the <5% of the errors undetected by the PBEDC scheme lead to negligible impact (no prediction changes or accuracy degradation).

This experiment reveals that the existing self-test scheme of [8] is not efficient for detecting single-bit errors and it also confirms the superiority of the proposed PBEDC scheme. Moreover, in the proposed scheme, the addition of such a small number of PBEDC samples to the standard inference can be very effective in detecting most of the single-bit errors.

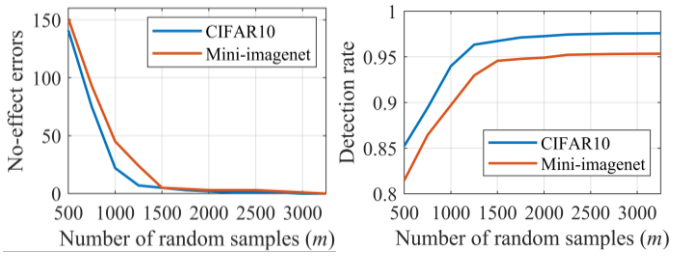
In terms of computational overhead, the insertion of 10 PBEDC samples for every 10,000 inference samples appears to be a feasible strategy, resulting in a 0.1% overhead for the additional samples. This is lower than for the existing self-test in [8] which requires 30 samples, i.e., with a reduction of 66%; but it achieves a finer detection not limited to errors leading to prediction changes. If considering the total number of parameters in the model, the overhead in memory can likely be further reduced. For example, the "RN50" CLIP network has 63 million parameters in the text encoder and 24 million parameters in the image encoder; storage of the PBEDC error detection during inference is reduced to $\frac{0.06MB}{174MB+61.44MB+0.06MB} \approx 0.026\%$. By the complexity analysis in Section III.E, the overhead does not increase proportionally to network size, thereby highlighting that the PBEDC offers significant advantages for applications involving large-scale networks.

C. *Row size of synthetic detection matrix: Random samples*

The size of the synthetic detection matrix is important for selecting the PBEDC samples and then the error detection performance. As discussed in Section III.C, two variables determine the size of this matrix, including its row size m and column size n. This subsection discusses the selection of m, which is the number of random samples in the synthetic detection matrix.

Intuitively, increasing the number of random samples m in the synthetic detection matrix benefits the PBEDC selection, because it is more likely to obtain representative samples that can achieve higher error detection rates for general evaluation sets. However, the generation of a synthetic detection matrix with a larger value of m significantly increases the complexity of simulation and the selection of PBEDC samples; its marginal effect may also only lead to eventually negligible benefits. On the other hand, the value of m cannot be very small either, because the random samples may not be enough for selecting the representative PBEDC samples due to some "no-effect" errors (which are defined by the errors that cause no change to the check node for all samples). Therefore, a proper value of m should be considered when its benefits start to get saturated.

For the applications considered in this paper, the impact of the number of random samples in the synthetic detection matrix is evaluated next. To do this, a fixed number of $n = 1000$ is selected for the matrix and the value range of m is considered from 500 to 3500. In each case, 10 PBEDC samples are selected by using the *max overlapped coverage* strategy because it achieves a better error detection performance as evaluated previously; the general evaluation set (different samples/errors) is utilized in this experiment.



(a) (b)

Fig. 9. Impact of the number of random samples (m) in the synthetic detection matrix on: (a) the number of "no-effect errors"; (b) error detection rate.

The simulation results in Fig. 9 (a) show that the use of more random samples in the synthetic detection matrix effectively reduces the number of "no-effect" errors; this is the case until the number of "no-effect" errors becomes very small. For example, the number of these errors reduces from 140 (150) to 5 (4) when m increases from 500 to 1200 (1500) samples for the dataset CIFAR10 (Mini-imagenet). After this, the reduction of "no-effect" errors becomes very slow. Note that this number may not eventually reduce to 0 in some cases, because some errors can be injected into a part of CLIP that does not propagate to any check nodes (as discussed previously in Section III.A).

Furthermore, Fig. 9 (b) shows that with an increasing number of samples in the synthetic detection matrix, the error detection rate first increases significantly but then, its change is rather small when m is larger than 1200 (1500) for CIFAR10 (Mini-imagenet). The trend is expected because a large improvement in error detection performance should occur only when the PBEDC samples are more representative and the number of "no-effect" errors is rapidly reduced; this is achieved when the number of random samples is increased starting from a small value of m (Fig. 9 (a)).

Overall, considering the tradeoff between error detection performance and complexity of matrix generation, the number of samples in the synthetic detection matrix should be at least selected as the value when the number of "no-effect" errors is close to 0; this guarantees most of the benefits, because further increments in samples only lead to marginal effects. Moreover, this experiment shows that only a limited number of random samples in the synthetic detection matrix is required in the proposed PBEDC scheme to achieve satisfactory error detection performance.

D. *Column size of synthetic detection matrix: Injected errors*

Next, the selection of the number of injected errors n in the synthetic detection matrix is analyzed. Intuitively, a larger n should lead to a better selection of PBEDC samples, because they can cover more possible error cases; however, in this argument there is a diminishing marginal benefit, because it is impossible to detect an unlimited number of errors when considering the complexity of error injection experiments. Therefore, like m, the value of n should also be selected within the range exceeding which the benefits of increasing n get saturated.

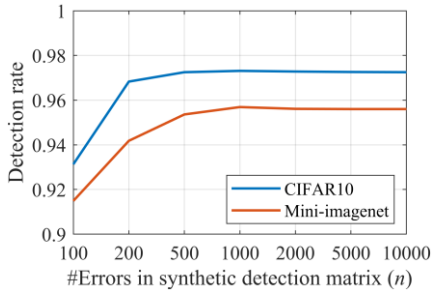


Fig. 10. Error detection rate of the proposed scheme using different number of injected errors (n) in the synthetic detection matrix.

The same evaluation method as in Section IV.C is conducted next; however, in this case, the value m of the synthetic detection matrix is fixed as 1200 (1500) for dataset CIFAR10 (Mini-imagenet), and different values of n are considered. The evaluated error detection rate results for different values of n are given in Fig. 10. The results show that for both datasets, a good number of n falls into the range of 500 to 1000, exceeding which the error detection performance does not improve. Therefore, there is no benefit by further increasing n; in other words, a small number of injected errors for establishing the synthetic detection matrix is representative enough to reflect the changes of the internal signals in CLIP as a large-scale NN.

E. Error correction performance

To evaluate the error correction performance of the proposed PBEDC scheme, single-bit errors have been injected as previously. The PBEDC samples are selected by the *max overlapped coverage* strategy of the synthetic detection matrix with a size of 1200×1000 for dataset CIFAR10 and 1500×1000 for dataset Mini-imagenet as suggested in the previous subsections.

In this paper, the simulation results are averaged over 100 runs; this shows that the error correction rate results are identical to the plots in Fig. 8 (i.e., > 95% errors for both datasets) because all detected single-bit errors can be corrected without any mis-corrections. Such results are also expected, because the probability that a mis-correction accidentally eliminates the perturbation is negligible.

As illustrated in Fig. 6, the bit-flipping correction process of the proposed scheme starts from checking position 1 (the sign bit) to position 16 (the last mantissa bit). Therefore, it is of interest to evaluate the distribution of corrected errors on all bit positions; this is performed for dataset CIFAR10 as an example. As per the results given in Fig. 11, when focusing on the internal softmax changes, errors on all bits can be handled generally. However, if only the prediction changes are monitored (like the method used in the existing self-test scheme of [8]), the correctable errors are shown to be concentrated in the MSBs. Even though using predictions as check nodes generally requires fewer trials to accomplish complete correction, the significantly smaller number of detected errors makes it not feasible for single-bit error correction.

Since only one PBEDC sample is required to locate the erroneous bit in the proposed scheme, even in the worst case (having to check all 16 bits), the error correction overhead is acceptable in a standard inference process. Compared to

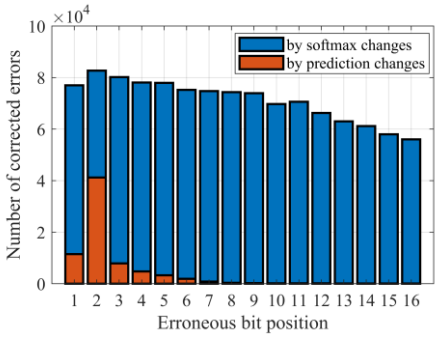


Fig. 11. Distribution of the corrected errors on all bit positions by applying softmax/prediction changes as check nodes.

traditional Single Error Correction (SEC) methods, that typically require five additional bits to protect and correct single bit errors on 16-bit data, the PBEDC approach significantly reduces the storage overhead by requiring only one parity bit. While PBEDC involves a small number of test samples, the memory overhead is negligible compared to the bit-wise protection for the entire network. Therefore, approximately 80% of memory savings is achieved by using a single parity bit in the proposed scheme versus five bits in an SEC code. This makes PBEDC a highly efficient and scalable solution, particularly suited for large-scale neural networks with constrained resources.

V. Conclusion and Future Work

This paper has proposed a Perturbation-Based Error Detection and Correction (PBEDC) scheme for large-scale machine learning (ML) models to enhance dependability in operation. In PBEDC, intermediate signals (i.e., softmax vector) are used as check nodes, and a small set of representative PBEDC samples are applied to detect the perturbation caused by errors; this is achieved without adding redundancy in the implementation. According to the complexity analysis, the overhead of the proposed scheme is very small and does not scale up with the network size. Compared with existing self-test schemes that can only detect partial errors by using the final prediction as the check node, the proposed PBEDC achieves a superior error detection rate and can also detect single-bit errors in the weights. The detection is not limited to errors changing the final prediction and the number of required test samples is smaller. Moreover, PBEDC can also be used for error correction when combined with parity codes. For the selection of PBEDC samples, the size of the synthetic detection matrix has been extensively analyzed and simulated. The results on a CLIP network used for zero-shot learning have shown that only a limited number of random samples and injected errors used to establish the synthetic detection matrix, are required for satisfactory error tolerance; this size does not further increase with larger evaluation sets, thus suitable for large-scale ML applications.

Future work may study more advanced strategies to select PBEDC samples based on the synthetic detection matrix; for example, synthetically generated samples can be utilized instead of directly selecting them from the original datasets. Also, by combining the use of parity codes, the current

correction scheme can only deal with single-bit errors; enhanced correction schemes will likely be investigated for more comprehensive scenarios with a larger error rate. Considering that PBEDC relies on a predefined set of golden values, it may need to be updated when changing model parameters or data distribution. In dynamic environments, automated approaches such as online updating of golden values or modular re-computation should also be explored to further reduce the computational overhead.

REFERENCES

[1] D. Jiang, G. Li, C. Tan, L. Huang, Y. Sun, J. Kong, "Semantic segmentation for multiscale target based on object recognition using the improved Faster-RCNN model," *Future Generation Computer Systems*, vol. 123, pp. 94-104, Oct. 2021.

[2] L. M. Francis, N. Sreenath, "Live detection of text in the natural environment using convolutional neural network," *Future Generation Computer Systems*, vol. 98, pp. 444-455, Sept. 2019.

[3] R. Yu, F. Jin, Z. Qiao, Y. Yuan, G. Wang, "Multi-scale image–text matching network for scene and spatio-temporal images," *Future Generation Computer Systems*, vol. 142, pp. 292-300, Jan. 2023.

[4] D. S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," *IEEE Trans. on Neural Networks*, vol. 6, no. 2, pp. 446-456, March 1995.

[5] T. Haruhiko, M. Masahiko, K. Hidehiko and H. Terumine, "Enhancing both generalization and fault tolerance of multilayer neural networks," in *2007 International Joint Conference on Neural Networks*, pp. 1429-1433, Aug. 2007.

[6] S. S. Lee and J. S. Yang, "Value-aware parity insertion ECC for fault-tolerant deep neural network," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 724-729, Mar. 2022.

[7] A. Radford, et al. "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning*, PMLR, 2021.

[8] F. Meng, F. S. Hosseini, C. Yang, "A self-test framework for detecting fault-induced accuracy drop in neural network accelerators," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, Jan. 2021.

[9] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, M. Chen, " Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, 2022.

[10] F. Meng and C. Yang, "Exploring image selection for self-test in neural network accelerators," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Nicosia, Cyprus, pp. 345-350, July 2022.

[11] M. I. Jordan, T. M. Mitchell, "Machine learning: trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255-260, July 2015.

[12] T. Brown, B. Mann, N. Ryder, et al., "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877-1901, 2020.

[13] K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778, 2016.

[14] A. Vaswani, et al., "Attention is all you need, "*Advances in neural information processing systems, 30*, 2017.

[15] Y. Zhang, H. Jiang, Y. Miura, C. D. Manning, C. P. Langlotz, "Contrastive learning of medical visual representations from paired images and text." in *Machine Learning for Healthcare Conference*, pp. 2-25, Dec. 2020.

[16] IS Committee, "754–2008 IEEE standard for floating-point arithmetic," IEEE Computer Society Std, 2008.

[17] M. D. Emmerson and R. I. Damper, "Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application," *IEEE Trans. on Neural Networks*, vol. 4, no. 5, pp. 788-793, Sept. 1993.

[18] C.-T. Chiu, K. Mehrotra, C. K. Mohan and S. Ranka, "Robustness of feedforward neural networks," in *IEEE International Conference on Neural Networks*, San Francisco, CA, USA, 1993, pp. 783-788 vol.2.

[19] Ching-Tai Chin, K. Mehrotra, C. K. Mohan and S. Rankat, "Training techniques to obtain fault-tolerant neural networks," in *Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing*, Austin, TX, USA, 1994, pp. 360-369.

[20] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2004.

[21] S. Cavalieri and O. Mirabella, "A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks", *Neural Networks*, vol. 12, no. 1, pp. 91-106, Jan. 1999.

[22] J. Bernier, J. Ortega, I. Rojas and A. Prieto, "Improving the tolerance of multilayer perceptrons by minimizing the statistical sensitivity to weight deviations", *Neurocomputing*, vol. 31, no. 1, pp. 87-103, 2000.

[23] N. Kamiura, Y. Taniguchi, T. Isokawa and N. Matsui, "An improvement in weight-fault tolerance of feedforward neural networks," in *Proceedings 10th Asian Test Symposium*, pp. 359-364, Aug. 2001.

[24] T. Haruhiko, K. Hidehiko and H. Terumine, "Partially weight minimization approach for fault tolerant multilayer neural networks," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, vol.2, pp. 1092-1096, May 2002.

[25] C. H. Sequin and R. D. Clay, "Fault tolerance in artificial neural networks", in *Proceedings Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 1, pp. 703-708, Jun. 1990.

[26] B. S. Arad and A. El-Amawy, "On fault tolerant training of feedforward neural networks", *Neural Networks*, vol. 10, no. 3, pp. 539-553, 1997.

[27] L. M. Luza *et al*., "Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks," *IEEE Trans. on Emerging Topics in Computing*, vol. 10, no. 4, pp. 1867-1882, 1 Oct.-Dec. 2022.

[28] G. Li, S. K. S. Hari, M. Sullivan, et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1-12, 2017.

[29] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, 2009.

[30] S. Ravi, H. Larochelle, " Optimization as a model for few-shot learning," in *International conference on learning representations*, Apr. 2017.

[31] Z. Wang, F. Niknia, S. Liu, P. Reviriego, P. Montuschi and F. Lombardi, "Tolerance of siamese networks (SNs) to memory errors: analysis and design," *IEEE Trans. on Computers*, vol. 72, no. 4, pp. 1136-1149, 1 Apr. 2023.