Z. Gao et al., "Robustness against Faults in Configuration Memories of FPGA-based LLMs," in IEEE Transactions on Circuits and Systems for Artificial Intelligence, doi: 10.1109/TCASAI.2025.3552735.

Robustness against Faults in Configuration Memories of FPGA-based LLMs

Zhen Gao, *Senior Member, IEEE*, Lini Yuan, Jingyan Wang, Qiang Liu, Javier Conde, Pedro Reviriego, *Senior Member, IEEE*, Shulin Zeng, Yu Wang, *Fellow, IEEE*, Shanshan Liu, *Senior Member, IEEE*, Fabrizio Lombardi, *Life Fellow, IEEE*

***Abstract*—Large Language Models (LLMs) pose significant challenges in terms of speed and energy dissipation of AI systems. Dependability is a further important issue for LLM implementations; this is especially relevant for FPGAs that are vulnerable to soft errors in the configuration memory. Moreover, as current GPU based implementations are not energy efficient, there is interest in running LLMs on different technology platforms, such as FlightLLM (an FPGA based accelerator designed to run LLMs for energy efficiency). In this paper, we analyze and evaluate the robustness of FPGA-based LLMs against faults/errors in the configuration memories. For the evaluation, we first propose a PyTorch based fault injection simulator and based on the analysis of FlightLLM and we study its robustness against stuck-at faults on the configuration memory. Furthermore, we propose an efficient error detection technique based on a concurrent classifier. Evaluation results show that stuck-at errors on high bits of the logic units can dramatically degrade the LLM performance, and the proposed concurrent classifier can effectively detect errors with negligible complexity and overhead. Finally, a low-cost fault location scheme is proposed, so that the fault can be easily recovered by dynamic partial reconfiguration. The combination of the concurrent classifier error detection and fault location can be used to improve the robustness of a FPGA-based LLM efficiently, such as FlightLLM.**

***Index Terms*—Dependability, Large Language Models, FPGAs.**

I. Introduction

THE introduction of ChatGPT in 2022 has led to the rapid development and usage of Large Language Models (LLMs) in a variety of tasks [1]. For example, large companies such as Meta and Google have developed models like Llama [2] or Gemini [3]; smaller companies like OpenAI or Mistral have also proposed some models [4], [5].

LLMs typically have billions (in some cases even trillions) of parameters and their execution requires a significant amount of energy due to the large complexity [6]. This occurs at least in part because LLMs are commonly executed on GPUs that are not specifically optimized for the model. This issue has triggered the development of both Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) accelerators for LLMs [7]. An example is the FlightLLM accelerator for AMD FPGAs that can run for example the Llama-2-7B model at a fraction of the energy dissipation required by a commodity GPU [8]. FlightLLM uses a highly parallel logic optimized for the processing required in the LLM; it is designed to map the resources of the FPGA efficiently to achieve better performance and energy trade-offs than general purpose GPUs.

A further yet important requirement for computing systems is dependable operation, so in the presence of errors or faults [9]. Dependability is increasingly important as there is interest in using Machine Learning (ML) and Artificial Intelligence (AI) in safety-critical applications [10], [11], [12]. This has motivated recent works on the dependability of AI hardware [13], [14]. The impact of errors on a system depends on a number of factors, such as the underlying hardware and the algorithmic properties of the system; therefore, understanding the impact of errors and implementing fault tolerant mechanisms is a complex process, especially for large systems like LLMs.

For FlightLLM, the target hardware platform is SRAM based FPGAs (SRAM-FPGAs); these chips are susceptible to soft errors [15]. The most common event is a soft error on a memory bit; when the error occurs in a user memory, it can corrupt a parameter of the LLM, for example a weight or a variable used in the current inference, such as an element in the Key-Value cache. This is like other hardware platforms, for example an error on the memory of a GPU and has been previously studied [16]. However, when the bit error affects the FPGA configuration memory, it can change the logic implementation, so leading to a look-alike permanent fault until the FPGA is reconfigured [17]. The impact of errors on the configuration memory is commonly greater than errors on the user memory; also, its analysis is often more complex. This has been confirmed in some previous works such as [18] which has studied the impact of errors on the configuration

This work is supported by NSF of China (62171313) and is partially supported by the FUN4DATE (PID2022-136684OB-C22) and SMARTY (PCI2024-153434) projects funded by the Spanish Agencia Estatal de Investigación (AEI) 10.13039/501100011033 and by the Chips Act Joint Undertaking project SMARTY (Grant no. 101140087). *Corresponding author: S. Liu*.

Zhen Gao, Lini Yuan, Jingyan Wang and Qiang Liu are with the School of Electrical and Information Engineering, Tianjin University, Tianjin 300072, China. Email: {zgao, lnyuan, wjy_27, qiangliu}@tju.edu.cn.

Javier Conde and Pedro Reviriego are with ETSI de Telecomunicación, Universidad Politécnica de Madrid, Madrid 28040, Email: {javier.conde.diaz, pedro.reviriego}@upm.es.

Shulin Zeng and Yu Wang are with the School of Electronic Engineering, Tsinghua University, Beijing 100084, China (e-mail: zengsl18@mails.tsinghua.edu.cn; yu-wang@mail.tsinghua.edu.cn).

Shanshan Liu is with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, 611731, China. Email: ssliu@uestc.edu.cn.

Fabrizio Lombardi is with Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02215, USA. Email: lombardi@ece.neu.edu.

memory of an FPGA when implementing a convolutional neural network (CNN) accelerator.

Mitigation is also more complex for errors on the configuration memory. For errors on user memory, generic protection techniques such as checking the range of values of the internal nodes to detect excessively large values and their removal [19], can be used. However, the impact of errors on the configuration memory is more complex because not only large values are an issue. Therefore, simple mitigation schemes are not always applicable; an exception is Concurrent Classifier Error Detection (CCED) that uses a simple classifier to monitor some nodes of the computing system and detect errors when observing patterns that can correspond to an error [20]. As the impact of errors on the configuration memory can be large, their patterns on the nodes are also significant and thus, easy to identify.

In this paper, the impact of errors on FPGA-based LLMs such as FlightLLM is studied in detail; we then propose an error detection technique and a fault location scheme. A fault model that captures the main effects of errors on the configuration memory is used and extensive fault injection experiments are conducted. The results show that as expected, errors can have a large impact on system performance. Interestingly, the use of a concurrent classifier is very effective in detecting errors and can be implemented at low cost, and the combination of fault detection and location enables the efficient protection of FlightLLM. The main contributions of the paper are:

- For FPGA-based LLMs, we propose a PyTorch-based fault injection simulator using a hook mechanism in the software implementation; this is used to evaluate the robustness of FlightLLM.
- The robustness of FlightLLM is studied for the first time by providing initial results and insights on the impact of errors on FPGA-based LLM performance. We identify the reason for the different effect of faults on different MPUs of the accelerator.
- The use of a CCED approach for hardware faults in FPGA-based LLMs is investigated for the first time; evaluation results show its applicability and effectiveness in detecting errors at low overhead.
- An efficient fault location scheme is proposed based on the FlightLLM hardware features and the input features of common LLM tasks.

The rest of the paper is organized as follows. Section II summarizes the related work, including Transformer accelerators, FlightLLM, robustness and protection of ML systems (including FPGAs). Section III introduces the necessary preliminaries for the study of the impact of errors on FlightLLM, including two case studies (BERT and CLIP), the assumed fault model and the CCED approach. Section IV presents the proposed PyTorch-based fault injection simulator and the robustness evaluation results. Section V proposes a classifier for CCED, evaluates its effectiveness, and introduces an efficient fault location scheme. The paper ends in Section VI with the conclusion.

II. Related work

A. Accelerators for Transformers and LLMs

Several works have proposed customized architecture designs for Transformer models [21]-[28]. Some works put more emphasis on accelerating the sparse attention [21], [22], [25], [26], [29]; specialized architectures have been designed to fully utilize the pre-defined static attention pattern [21], [25] or the dynamically generated attention pattern [26]-[29]. Recently, the FFN-Attention Co-optimized Transformer Architecture (FACT) has pointed out the importance of compressing linear layers with mixed-precision quantization to reduce the latency [27]. However, these methods cannot accelerate the decode stage of the LLMs because they mainly focus on the prefill stage for discriminative models. DFX [23] has emphasized the acceleration of the decode stage of LLMs; however, it lacks the hardware support for model compression of the LLMs, hence making it hard to further expand model size or the maximum token size. To the best of the authors knowledge, FlightLLM is the first accelerator enabling efficient LLMs inference on a single FPGA [8]. Combined with compression techniques such as sparsification and quantization, FlightLLM can efficiently accelerate LLMs and reduce the inference overhead.

B. FlightLLM

Based on an evaluation using Xilinx Alveo U280 FPGA, FlightLLM achieves 6.0× higher energy efficiency and 1.8× better cost efficiency against commercial GPUs (e.g., NVIDIA V100S) on modern LLMs (e.g., LLaMA2-7B) using vLLM and SmoothQuant under the batch size of one [8]. The main innovations of FlightLLM for such high efficiency include: 1) a configurable sparse DSP chain to support different sparsity patterns with high computation efficiency; 2) an always-on-chip decode scheme to boost memory bandwidth with mixed-precision support.

As shown in Fig. 1, the overall hardware architecture of FlightLLM includes a task scheduler, a memory controller, and multiple computing cores. The task scheduler assigns tasks to different cores and controls the data flow. Each core includes the unified Matrix Processing Engine (MPE), the Memory Management Unit (MMU), the Special Function Unit (SFU) and the Instruction Scheduler. The instruction scheduler decodes the instructions and schedules different hardware units to perform computations. The MPE handles all matrix operations in LLMs and utilizes the configurable sparse DSP chain to reduce the hardware overhead. The MMU reduces memory access overheads by designing customized quantization units for low-bit mixed-precision and improving data placement for off-chip memory. The SFU handles miscellaneous operations (e.g., Softmax, etc.) in addition to matrix processing operations.

Fig. 2 shows the MPE; in Fig. 2(a), the MPE includes N Matrix Processing Units (MPUs), and each MPU includes M vector processing units (VPUs). In addition, weight and activation buffers are used to store the weights and input activations, respectively; the global buffers store the output

activations of the MPE. The VPU is the basic component in the MPE, it performs the dot product of two vectors. As shown in Fig. 2(b), a VPU consists of *K* DSP groups, and each group has two DSP48 cores. A configurable sparse DSP chain supports sparse matrix operations by adding the Sparse Mux, Reduction Node and Overflow Adjust unit. The reader should refer to [8] for more details about these units. To improve efficiency, all inputs of the VPUs (weights and activations) are quantized to integers in FlightLLM.

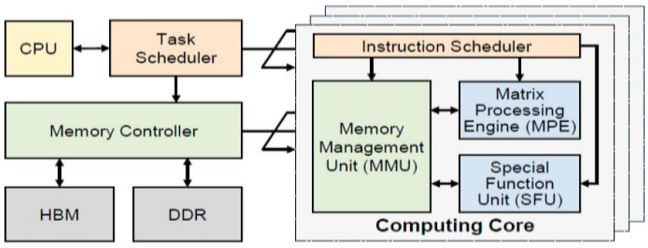
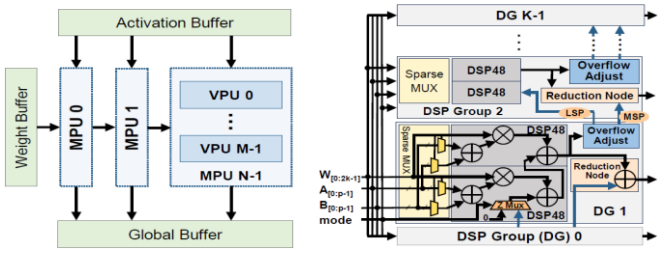


Fig. 1. The FlightLLM accelerator [8]



(a) MPE with multiple MPUs (b) VPU with multiple DSP Groups

Fig. 2. The Matrix Processing Engine (MPE) [8]

C. Robustness of FPGA implemented ML systems

In SRAM-FPGA based ML accelerators, user memories mostly hold the weights and feature maps (or activations), while configuration memories define the logic functions of the circuit. Many studies have evaluated the impact of errors on both types of memories, but most of them focus on small scale models, such as CNNs. The results of [30] and [31] have revealed that errors on a small portion of weights (e.g., 10^{-4}) do not degrade task performance, and memories for feature maps are significantly more tolerant to errors. The results of [32] have proved that the impact of errors on user memories can be reduced by normalization, nonlinear activation and network compression techniques. However, errors on configuration memories have more severe impacts [33]. For example, radiation experiments in [34] have revealed that errors on some layers of a CNN may cause severe performance degradation; [35] has shown that errors on about 20% of the configuration memory bits cause wrong classifications for a CNN based traffic sign recognition; [36] has revealed that larger parallelism increases the radiation sensitivity of the FPGA accelerator. [37] and [38] have performed a systematic evaluation of FPGA-based CNN accelerators to errors on configuration memories based on fault injection experiments and have showed that errors on memory access logic and control unit may cause severe system failures. Furthermore, [39] has proposed models for the effect of errors on configuration memories of CNN accelerators and found that a large portion of errors on the computing module introduce stuck-at equivalent faults on the input or output of the multipliers and adders.

In recent years, researchers have started to study the impact of errors on Transformers and LLMs, but most works have focused on the errors in user memories. For example, [16] and [40] have studied the robustness of Bidirectional Encoder Representations from Transformers (BERT) to soft errors on the floating-point and quantized weights, respectively; [41] has studied the effect of errors on the weights of Stable Diffusion, while [42] has studied the impact of errors on the embeddings in typical transformer models for natural language processing (NLP) applications. All these works show that errors on some critical bits cause a severe degradation of the task performance. Finally, [43] is the first work that has evaluated the impact of errors on a widely applied open source LLM (Mistral-7B) and has showed the intrinsic tolerance of Mistral-7B to errors on the quantized weights. However, as far as the authors' best knowledge is concerned, there is no research discussing the robustness of FPGA-based Transformers or LLMs against faults on the configuration memories.

D. Protection schemes for ML systems

Convolution or matrix-vector multiplication (MVM) are the most complex operations in deep neural networks, so many researchers have proposed Algorithm-Based Fault Tolerance (ABFT) schemes to protect the computing core from errors [30],[44]-[47]. In general, the processing of the Processing Element (PE) array is formulated as parallel MVMs in these schemes. Hence, coding can be applied to introduce redundant columns or rows for detection and correction of an erroneous output of each PE. Furthermore, a more efficient "detect and replace" scheme has been proposed in [48] to protect CNN accelerators against errors on the configuration memory of FPGAs. The basic principle of this scheme is that if the faulty unit can be recovered (by replacement) in time, then errors introduced in the feature maps during the failure time, can be tolerated by CNN itself. Different from the above works, [49] and [50] have proposed the protection of a strong ML model by replacing it with an ensemble of several weak ML models. Due to the diversity in the weak models, the ensemble performance can be close or even higher than the strong model, and the failure of any weak model only introduces a small accuracy loss. This scheme can deal with system failures caused by errors in the configuration memory for the control logic in FPGA accelerators [37], [38].

All of the above schemes can only guarantee the ML performance when the hardware faults occur, but they cannot remove the faults. In practice, we can apply periodic scrubbing to recover from faults on the configuration memory of FPGAs [51], [52]. However, this technique affects the operation of the logic, and it also brings additional power consumption. In this paper, we propose efficient error detection and fault location schemes, so that we can initiate partial reconfiguration only for a specific configuration frame in which the fault is located.

III. PRELIMINARIES

The main objectives of this paper are to acquire initial understandings of the impact of soft errors in the configuration memory of FPGA based LLM accelerators and then investigate efficient error detection and location designs. Therefore, in this section all necessary preliminaries for achieving these objectives are presented.

- Since the robustness of the model is usually measured by the impact of the faults on the performance of specific tasks, we introduce two popular Transformer models as case studies in sub-section A.
- As the key for evaluating the impact of errors, the fault model is discussed in sub-section B, which is also the basis for the fault injection simulator (as introduced in detail in Section IV.B).
- We apply an existing error detection method, CCED for fault detection in Section IV, so the general principle is first introduced in sub-section C.

In summary, we inject faults on two typical Transformer models based on the fault model, and the erroneous outputs are used as input to the CCED for error detection.

A. Case Studies: BERT and CLIP

In this paper, BERT and CLIP are taken as case studies to evaluate the impact of errors on FlightLLM. We choose these models based on two considerations:

- These are popular Transformer models; BERT is a text encoder and is widely applied in NLP applications and CLIP also includes an image encoder that is widely applied in computer vision (CV) and multi-modal applications. NLP, CV and multi-modal cover the main application fields of Transformer and LLMs.
- Different Transformer models apply similar structures as in BERT and CLIP (e.g. stacks of Transformer layers composed of self-attention and feed forward networks), and FlightLLM implements different Transformer models with the same hardware architecture (e.g. using MPE for all matrix-matrix multiplications (MMMs)).

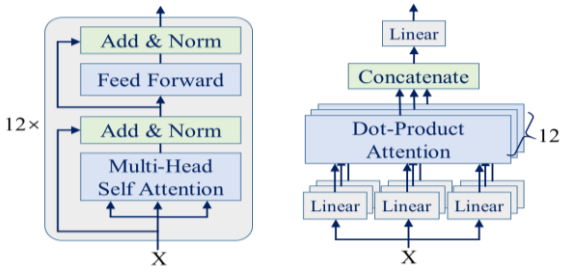
Therefore, errors would have similar effects on other Transformer and LLM models as in BERT and CLIP, and the evaluation results for the case studies also provides general findings that are valuable for other Transformer models running on FlightLLM.

In this section, we briefly introduce BERT and CLIP and their corresponding tasks, and the fault injection evaluation results are provided in Section IV.C.

1) BERT based sentence emotion classification

BERT has been proposed by Google in 2017 for NLP applications [53]. As shown in Fig. 3(a), BERT consists of 12 Transformer encoders, and each encoder includes the Multi-Head Self Attention (MHSA) layer and the Feed Forward Network (FFN). Input X is the embedding matrix for the input text or the activation output of the previous encoder. As shown in Fig. 3(b), MHSA first performs linear operations to generate the Key, Query and Value matrices as K = X*Wk+Bk, Q = X*Wq+Bq and V = X*Wv+Bv, respectively, for 12 heads (W is the weight matrix and B is the bias vector), and then calculates the attention for the *h*-th head as $A_h(X) = Softmax(Q*K^T)*V$. After concatenating all heads as $A_C = [A_1, A_2, \ldots, A_{12}]$, a linear operation is performed to get $MHSA(X) = A_C*Wo + Bo$. After residual addition and normalization (results denoted as Y), two linear operations are performed in FFN to calculate the output of the encoder as $Z = W_{F2}* Act(Y*W_{F1}+B_{F1})+B_{F2}$, where Act() is the non-linear activation function, which becomes the input of the next encoder layer.

In this paper, we use BERT for sentence emotion classification task. Specifically, an input sentence (with the [CLS] token at the beginning) is transformed to a semantic matrix by BERT, and the first row of the output matrix of the last layer is used to determine one of six emotions: 'joy', 'sadness', 'anger', 'fear', 'love' and 'surprise'. We use the fine-tuned model from Huggingface [54] for the evaluation in this paper; with a test set of 2000 sentences, the accuracy of the BERT based sentence emotion classification is 92.1% in the error free case.



(a) General structure (b) Structure of MHSA

Fig. 3. Model structure of BERT

2) CLIP based image classification

CLIP is a multimodal Transformer model proposed by OpenAI in 2020 for zero short image classification [55]. As shown in Fig. 4, CLIP consists of a text encoder and an image encoder. The text encoder is an encoder-only Transformer that transforms the name of a class to a semantic vector, and the image encoder (based on the modified ResNet or Vision Transformer (ViT)) transforms an image to a semantic vector. Then the class that has the highest similarity in the semantic space is selected as the classification result. For both encoders, the semantic vector is the first row of the final output matrix, corresponding to the [CLS] token at the beginning of the inputs.

In this study, ViT-L/14 is used as the image encoder, and CIFAR-100 is used to evaluate the accuracy of the image classification. The test set consists of 10000 images with 100 images per category. The open-source implementation of the CLIP model [56] is used in this paper for evaluation, and the classification accuracy is 77.4% in the error free case.

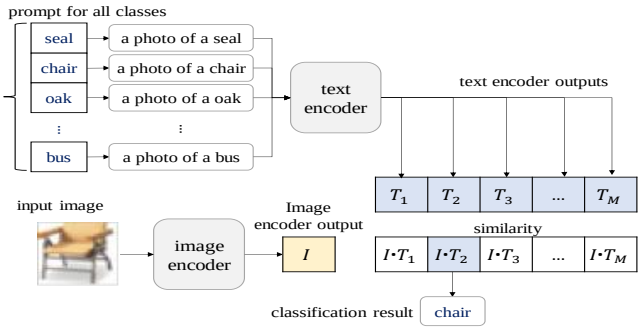


Fig. 4. CLIP and the image classification process [55].

B. *Fault Injection and Fault Model*

The common approach for a robust evaluation is to conduct fault injection (FI) experiments directly on the FPGA running the LLMs. For example, many works have proposed to apply SEM IP for SoC FPGAs or Microblaze soft processor to access a specific frame of the configuration memory through ICAP [18], [57], [58]. However, FlightLLM is currently implemented using the Xilinx Runtime (XRT) framework on the Xilinx Alveo U280 FPGA [8]. The XRT framework facilitates the development of an accelerator by providing a standardized software interface to the FPGA [59], but it includes the soft processor and the ICAP in a shell on the FPGA that is not accessible from the user space. Therefore, currently available FI tools cannot be used directly to evaluate the robustness of FlightLLM. To address this limitation, this paper proposes to use a software simulator for fault injection experiments; hence, the first important step is to determine the fault model.

Recent works on error modeling for FPGA-based CNN accelerators have shown that stuck-at faults on the inputs and outputs of the DSP array and adder tree account for a large portion of the effects of errors on the configuration memory [39]. Since MPE is the main computing module in FlightLLM, in this paper we focus on the MPE for the assessment of the robustness and leave the study of other modules in future work. As shown in Fig. 2, the MPE is also implemented using a DSP array and adder tree, so the effect of faults on the configuration memory of MPE would be like as for the PE array in a CNN. Therefore, we take the stuck-at faults as an approximate model to evaluate the impact of errors on FlightLLM and consider both stuck-at-0 and stuck-at-1 faults. Different from the convolution operation in CNNs, the data input and weight input are equivalent for MMM in Transformers, and the outputs of an MMM are used as the inputs to the other MMM in many cases (e.g. the MHSA or FFN shown in subsection A). Therefore, we can focus on the stuck-at faults at the output of MMM to approximately cover the effect of faults on the data input, weight and the results.

As shown in Fig. 2(a), MPE consists of an array of VPUs. Therefore, we assume one bit of the output of a VPU in an MPU is fixed at 0 or 1; we believe that this model captures the most significant effects of errors on the configuration memory of FlightLLM. We have implemented a novel PyTorch based simulator for stuck-at fault injection on the VPU; moreover, the erroneous outputs of the Transformer models are used as input of the concurrent error detection classifier. The details of the fault injection simulator are introduced in Section IV.B.

C. *Concurrent Classifier Error Detection (CCED)*

Since errors on the configuration memory can be removed by just reconfiguring the FPGA, error detection that can trigger a reconfiguration is sufficient for protecting FPGA-based LLM accelerators. Therefore, the CCED approach (as discussed in Section V.A) is considered in this paper.

A general CCED scheme to detect errors in large ML systems has been proposed in [20]. This scheme monitors several nodes of the system and generates a dataset with values for error-free executions and for executions on which an error has been injected; then, this dataset is used to train a classifier to detect errors. Finally, the classifier is implemented and run concurrently with the main machine learning system to detect errors (Fig. 5). The use of CCED is attractive because it can be implemented without requiring changes in FlightLLM and relevant errors on the configuration memory are expected to have a large impact and therefore, they should be detectable by a classifier. Once an error is detected, we can first locate the fault (as discussed in Section V.C) and then recover it by dynamic partial reconfiguration at a very low cost.

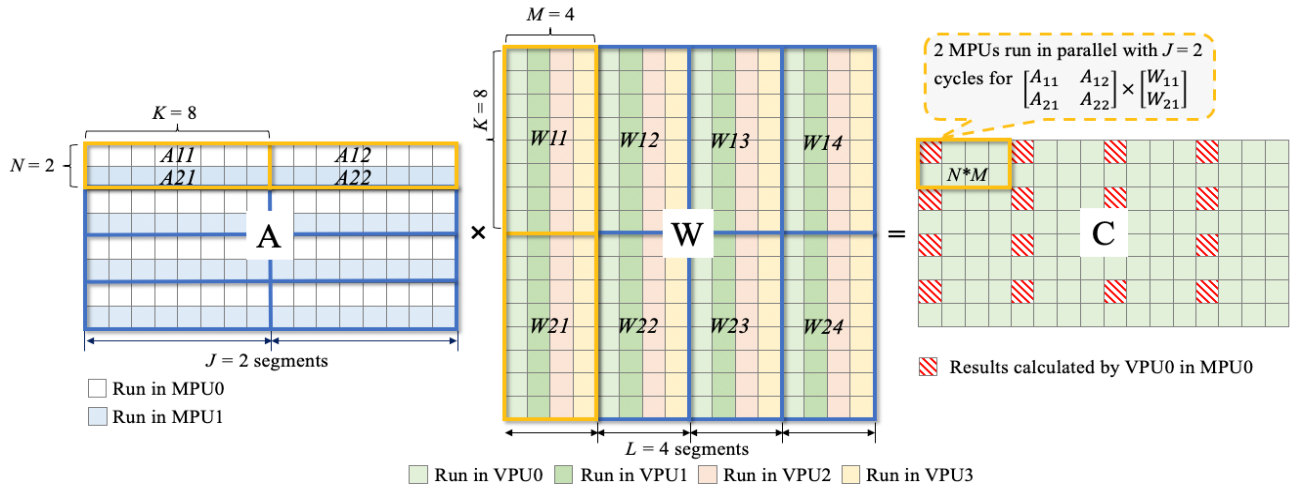


Fig. 6. The computational model of an MMM in the MPE by taking Nr = 8, P = 16, Nc = 16, N = 2, M = 4, K = 8 as an example (note that all shadowed results can be corrupted by a single error in one VPU0)

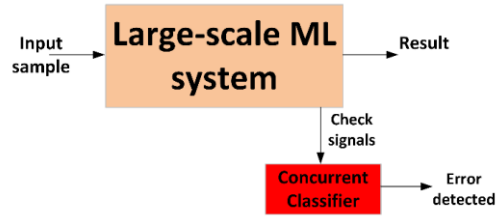


Fig. 5. Concurrent Classifier Error Detection (CCED) scheme [20].

IV. Fault Injection Simulation and Results

The impact of a stuck-at fault on the output of a VPU is related to the structure of the MPE; therefore, this section first introduces the parallel computing architecture of the MPU and then proposes the simulator for the faults in the MPE hardware with the Python implementation. Finally, the impact of the stuck-at faults on the task performance is presented.

A. Computational Model of MPE

All linear operations in BERT and CLIP are performed by the MPE sequentially. We assume that the MPE is used for the MMM between activations and weights as A*W = C, where A $\in Z^{Nr\times P}$, W $\in Z^{P\times Nc}$ and C $\in Z^{Nr\times Nc}$. The computing process is shown in Algorithm 1, where N is the number of MPUs, *M* is the number of VPUs in an MPU and *K* is the number of multipliers in a VPU. An MPU works for the items in a row of C. A VPU performs (1**K*)*(*K**1) vector multiplications in one cycle. Each row of A is divided into J segments (J = P/K), and the j-th segment for the *r*-th row of A is denoted as A_{rj} (r = 1, 2, …, Nr, j = 1, 2, …, J). The weight matrix W is divided into J*L small matrices $W_{jl} \in Z^{K\times M}$ (j = 1, 2, …, *J*; l = 1, 2, …, *L*), where *L* = *Nc*/*M*. In a MPU, one A_{rj} is broadcasted to *M* VPUs so that the (1**K*)*(*K***M*) multiplication for A_{rj}*W_{jl} can be performed in one cycle. Therefore, all MPUs run in parallel for the MMM of (M*K)*(K*N)→(M,N) in *J* cycles.

Fig. 6 shows the computational method of an MMM in the MPE with an example of (8*16)*(16*16)→(8,16) (*Nr* = 8, *P* = 16, *Nc* = 16) using a MPE with 2 MPUs (*N* = 2), 4 VPUs in a MPU (*M* = 4) and 8 multipliers in a VPU (*K* = 8). The shadowed blocks in the result matrix show the values calculated by VPU0 in MPU0. If stuck-at faults occur in this VPU, then these values would be corrupted.

Algorithm 1: A*W = C: (Nr*P)*(P*Nc)→(Nr,Nc) in MPE

```
for r = 1:N:Nr //split rows of A into groups of N
  for c = 1:M:Nc //split columns of W into groups of M
    //VPUs in all MPUs run in parallel for (N*K)*(K*M)→(N,M)
    for p = 1:K:P //split columns of A into groups of K
      for n = r:1:r+N // N MPUs run in parallel for N rows
        for m = c:1:c+M // M VPUs run in parallel
          for k = p:1:p+K // K multiplications in a VPU
            {C(n,m) += A(n,k)*W(k,m)}
```

B. PyTorch based Fault Injection Simulator

In the open source PyTorch code for Transformer models from Huggingface, most of the key operations are defined as a class of *module*. For each module type operation, two hook functions are provided to the user, *forward_ pre_hook*() and a *forward_hook*(). The user can define *forward_pre_hook*() to change the input before the execution of the operation, and define *forward_hook*() to change the output after the execution of the operation. Therefore, we can use *forward_pre_hook*() to inject faults in the input data, and use *forward_hook*() to change the output to simulate the effect of faults on the function of the operation.

In the code for BERT and CLIP, the main operations are defined as a module, including among others the MMM, activation and normalization. As the most frequent operation in FlightLLM, the MMMs in the MHSA and FFN are implemented based on the VPU array in the MPE, and the stuck-at fault on the output of a MMM is the input for the next MMM. Therefore in this paper, we only inject stuck-at faults at the output of a VPU using *forward_hook*(). Note that this method can be used for fault injection of all module type operations.

After determining the position of the stuck-at faults (e.g. the *i*-th bit of the *m*-th VPU in the *n*-th MPU), the fault injection procedure is executed (Fig. 7). The following procedures are then utilized:

1) Since the format for the weights and activations in the original PyTorch implementation is Float16 (FP16), they are first quantized to 8-bit integers (INT8) as used in the MPE; so, the SmoothQuant scheme [57] is applied (as used for FlightLLM).
2) The computation related to the *m*-th VPU is repeated with the corresponding INT8 weights and activations; the output is obtained for the VPU (O_{VPU}).
3) The error caused by the stuck-at 0 (SA0) or stuck-at 1 (SA1) faults on the *i*-th bit of O_{VPU} is established based on Algorithm 2 (*i* = [0, 15], the 15-th bit is the sign bit). As for the INT16 result, O_{VPU} is represented as a complement code, the error is also related to the sign of O_{VPU}.
4) The error on the value of the result matrix *C* related to the *m*-th VPU is accumulated *J* times (*J* = *P*/*K*).
5) The INT16 accumulated error is changed back to a FP16 value, and then used to change the corresponding values in *C*.

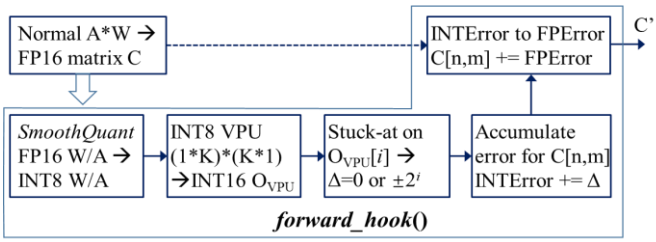


Fig. 7. Procedure for stuck-at injection on the output of a VPU

Algorithm 2: Error by a Stuck-at fault on the i-th bit of O_{VPU}

// **SA0**	// **SA1**
if O_{VPU}[i] = 0: Δ = 0;	if O_{VPU}[i] = 1: Δ = 0;
else	else
if $O_{VPU} \geqslant 0$: Δ = -2^i;	if $O_{VPU} \geqslant 0$: Δ = 2^i;

else $\Delta = 2^i$;	else $\Delta = -2^i$;

C. Simulation Setup and Results

In the current version of FlightLLM, there are 3 MPUs (N = 3), 32 VPUs in a MPU (M = 32) and 32 multipliers in a VPU (K = 32); so, we also perform MMM in the PyTorch implementation with this configuration. Considering that the faults on different VPUs in different MPUs may have different effects, we inject faults on 6 VPUs in MPU0 (n = 0, m = 0, 6, 12, 18, 24, 30), 6 VPUs in MPU1 (n = 1, m = 1, 7, 13, 19, 25, 31) and 5 VPUs in MPU2 (n = 2, m = 2, 8, 14, 20, 26). For each VPU, we inject SA0 or SA1 faults on a different bit position (i = [0, 15]). Based on the experiment results, we found that faults at different VPUs in the same MPU have similar impact to the performance. This is reasonable because as shown in Fig. 6, different VPUs compute different items of the same row of the output matrix. However, the impact of faults on different bit positions and MPUs is very different.

Fig. 8 shows the accuracy for an SA0 or SA1 on different bit positions for the BERT based sentence emotion classification task. Each point in the figure is averaged over 17 VPUs in 3 MPUs (6+6+5). We can draw two observations from this figure:

- SA faults on higher bits cause a larger accuracy loss, only errors on the 5 or 6 (for SA0 or SA1) most significant bits (MSBs) cause a significant loss.
- The accuracy loss by SA1 faults is larger than for SA0 faults.

The first observation is easier to understand because faults on higher bits cause a larger change in the result. The second observation can be explained based on the amplitude difference of the results modified by SA0 and SA1. For the sign bit (i = 15), the change of amplitude by SA0 or SA1 is the same, (i.e. 2^{15}), so the impact on the accuracy is also the same. However, for other bits, a SA1 (SA0) increases the positive (negative) value and decreases the negative (positive) value. This makes the results modified by SA0 closer to 0 than those by SA1; therefore, SA0 faults cause a smaller accuracy loss than SA1 faults.

Fig. 9 shows the effect of SA0 or SA1 faults on different MPUs for 5/6 MSBs, in which each point is averaged over the 6/5 VPUs in the corresponding MPU. The impact of faults on MPU1 and MPU2 is similar, however faults on MPU0 introduce a higher accuracy loss. This occurs because as shown in Fig. 6 and Algorithm 1, MPU0 performs the calculation of the first row of the output matrix. As introduced in Section III.A, the first row of the input to BERT corresponds to the [CLS] token, and the first row of the final output matrix is used for emotion classification. Therefore, the faults on MPU0 directly affect the classification result.

Fig. 10 and Fig. 11 show the effect of SA faults on different bit positions and different MPUs for CLIP. The main observations for BERT still hold for CLIP; the significant difference is that the number of critical bits for MPU0 is 7 and 8 for SA0 and SA1, respectively, but for MPU1 and MPU2, it is 4 for both SA0 and SA1.

Based on the above results, the fraction of faults that cause a significant performance degradation can be estimated. Since we inject all 16-bit positions of each of the 32 VPUs in each MPU, the fraction for BERT is calculated as (5/16+6/16)/2 = 34.4%, and for CLIP as (7+4+4)/16/3 + (8+4+4)/16/3)/2 = 32.3%. Therefore, although the fraction of critical faults is related to the model and task, it is approximately around 1/3 of all stuck-at faults in both cases, so showing the intrinsic robustness of FlightLLM.

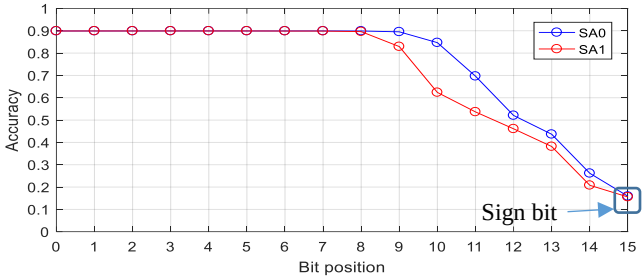


Fig. 8. Effect of Stuck-at faults on different bit positions for BERT

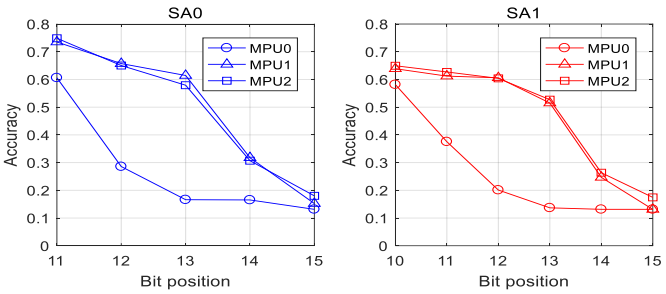


Fig. 9. Effect of SA faults on different MPUs for BERT (for 5/6 MSBs)

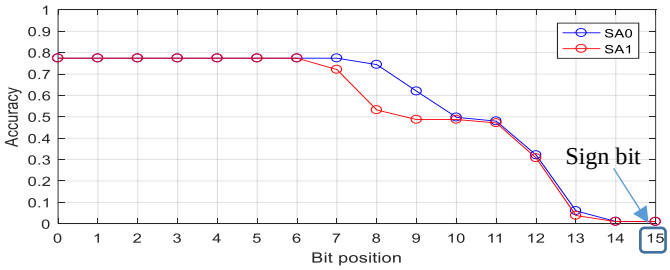


Fig. 10. Effect of Stuck-at faults on different bit positions for CLIP

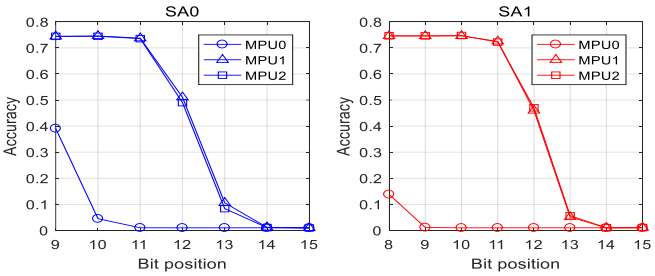


Fig. 11. Effect of SA faults on different MPUs for CLIP (for 7/8 MSBs)

V. Proposed CCED Designs and Fault Location

The use of a concurrent classifier has been proposed in [20] to detect errors on the parameters of large-scale machine learning systems. This approach is well suited to FPGA-based LLMs because it can be implemented without a significant modification of the LLM accelerator. However, errors on the configuration memory are completely different from errors on the model parameters; hence, the effectiveness of a concurrent classifier cannot be taken for granted and a detailed analysis

and evaluation are needed to confirm its applicability for FPGA based LLM accelerators.

This section explores the use of CCED approach [20] to detect errors on the configuration memory bits of a FPGA-based LLM, with particular emphasis on FlightLLM. Initially, the design of the concurrent classifiers is discussed for BERT and CLIP, then their effectiveness is evaluated. Finally, a fault location scheme is proposed for low-cost recovery when CCED report a fault.

A. Concurrent Classifier Design

In this scheme, the objective is to train a binary classifier that can detect errors in the LLM based on its predictions, but without affecting system performance. To make this process efficient, few requirements must be enforced:

- It must be a fast model (so operating significantly faster than the LLM prediction speed);
- it must be less complex in hardware (for example, memory resources are reserved for the LLM);
- it must avoid reconfiguring the FPGA when unnecessary.

The first two conditions can be achieved with models like decision trees [61], random forest [62], or naive Bayes [63]. For the third condition, it is necessary to ensure that the models have a low false positive rate. To achieve this objective, the decision threshold can be adjusted to heavily penalize false positives. However, balancing the threshold to reduce the number of false positives leads to a significant increase in false negatives that are also not desirable. A valid alternative for the proposed scheme is to consider that there is an error in the LLM when two consecutive errors are detected with the concurrent classifier. Otherwise, if the model detects an error in one prediction but not in the next, the first prediction is considered a false positive. As per this condition, a 5% false positive (FP) rate from the classifier results in a system FP rate of approximately 0.25%.

This can be scaled to N_p predictions to further reduce the number of FPGA reconfigurations, while not be able to detect some errors. This problem can be modeled by an absorbing Markov chain; the initial state corresponds to the point in which the model has been reconfigured. The final state occurs when it is detected that the LLM has a failure (N consecutive errors are detected). Two scenarios may arise.

- First, there is no failure in the LLM, but partial re-configuration is triggered. This happens on average every $(FP^{-Np} - 1)/(1 - FP)$ LLM runs (Fig. 12, top) [64].
- Second, a failure occurred in the LLM; the average number of LLM runs until reaching the absorbing state (the partial reconfiguration) is given by $[(1 - FN)^{-Np} - 1]/[1 - (1 - FN)]$ (Fig. 12, bottom). As an example, with N_p = 2, FP = 5% and FN = 3%, the LLM is reconfigured in absence of a failure every 420 predictions on average; when a failure occurs, it takes on average 2.09 runs to detect it and reconfigure the LLM. A higher value of N reduces the effective FP, but it increases the number of LLM runs required to detect an error when it occurred.

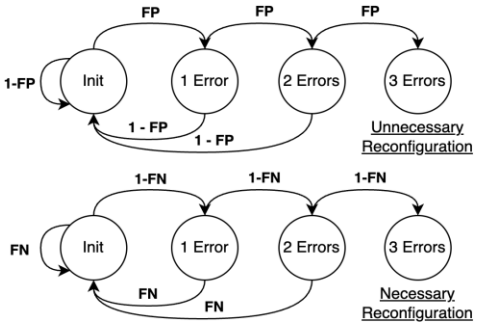


Fig. 12. Markov chain of the reconfiguration process when detecting N_p =3 consecutive errors in a LLM (with no error (top) and a bit error (bottom))

Next, the classifier is analyzed and assessed for BERT and CLIP as the two models considered in FlightLLM.

1) Concurrent classifier for BERT

The output of the BERT sentence emotion classifier is a vector of 6 scores, and these are the inputs to the binary classifier that attempts to detect errors in the classifier output. The dataset consists of 10,000 error-free samples. From these samples, 10,000 SA0 (2,000 for each of the 5 MSBs) and 12,000 SA1 (2,000 for each of the 6 MSBs) faults have been generated for each of the 6 VPUs in MPU0, 6 VPUs in MPU1 and 5 VPUs in MPU2, resulting in a total of 374,000 single-error samples. We divide the final dataset of 384,000 samples into 80% for training and 20% for evaluation. An analysis of the training dataset reveals the effect of errors in the distribution of the outputs of BERT. Fig. 13 shows the distribution of one sentiment output. In the absence of an error, the distribution has two peaks, i.e. around 1 (when it is the predicted emotion) and around 0 (when it is the non-predicted emotion). However, in the presence of errors in the BERT model, the distribution of results is more heterogeneous.

As per the above strategy, we combined the six outputs of BERT as the features of the concurrent classifier. We tested different combinations of models and hyper parameters to achieve a very high accuracy while keeping the false positive rate low to avoid unnecessary reconfigurations of the FPGA.

The best model obtained was a Balanced Random Forest Classifier [65] with 5 estimators. The balanced model was selected, because the dataset contains 1 error-free sample for every 31 errors; to limit overfitting, the largest depth of the model was adjusted to 5 levels. The least number of samples required to split a node is 2 and the least samples per leaf is 1 to maintain the model's variability, because overfitting is controlled by the number of trees and the maximum depth; Gini impurity is then used to measure the quality of splits because it is more robust for unbalanced classes than entropy; and by considering the square root of the total number of features for each split to reduce the correlation between trees. The remaining parameters are given at default settings of the Balanced Random Forest Classifier in imblearn [66]: the lowest weighted fraction of the total weights needed to reach a leaf node is 0, there is no limit in leaf nodes, it uses bootstrap samples with no replacement, no out-of-bag samples, no pruning, and no warm start.

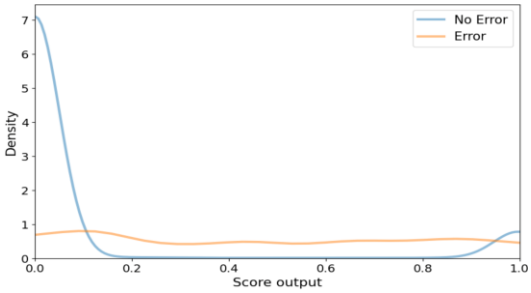


Fig. 13. Distribution of one output with error-free and single errors for BERT

2) Concurrent classifier for CLIP

The CLIP based image classifier on CIFAR-100 outputs a vector of 100 scores representing the probability of each class. The dataset consists of 10,000 error-free samples. From these samples, 14,000 SA0 (2,000 for each of the 7 MSBs) and 16,000 SA1 (2,000 for each of the 8 MSBs) faults have been generated for each VPU in MPU0, and 8,000 SA0 (2,000 for each of the 4 MSBs) and 8,000 SA1 (2,000 for each of the 4 MSBs) faults have been generated for each VPU in MPU1 and MPU2, resulting in a total of 356,000 single-error samples. Like in BERT, 80% of the data is used to train the model and 20% to evaluate it. The exploratory analysis of the training dataset (Fig. 14) reveals that errors affect in different ways the distribution of the outputs, some outputs are very sensitive (e.g., class 70), while others are not as much (e.g., class 17).

The best model obtained was a simple feed forward neural network designed for binary classification, with an architecture composed of five hidden layers and one output layer, and optimized to handle the complexity of the problem and prevent overfitting using the library Keras [67]. Again, there is a lack of balance between the classes (1 error free per 35.6 error samples). The minority class was oversampled using the Minority Over-sampling Technique (MOTE) in the training dataset to further address the class imbalance **¡Error! No se encuentra el origen de la referencia.**. The network starts with an input layer composed of 100 entries (all output scores obtained from CLIP), followed by a first hidden layer with 512 neurons and rectified linear units (ReLU) activation, to which Batch Normalization is applied to stabilize the activations. The second layer, also with 512 neurons and ReLU activation, incorporates Batch Normalization and a Dropout mechanism with a 50% rate to reduce the risk of overfitting. The third layer slightly reduces the size to 256 neurons and retains both normalization and a 50% Dropout. In the fourth layer, the size is further reduced to 128 neurons, repeating the pattern of ReLU activation, Batch Normalization, and Dropout. The fifth layer, with 64 neurons, uses the same activation and normalization, but without Dropout. Finally, the output layer has two neurons corresponding to the error and non-error classes and it employs a sigmoid activation to generate a probability distribution over the classes. The model was compiled using the Adam optimizer with a reduced learning rate of 0.0005, and the binary-crossentropy loss function, which is specific for binary tasks, with performance evaluated using accuracy as metric. It is trained for 30 epochs with a batch size of 32, using a validation set to monitor learning. Strategies such as Batch Normalization in all layers and Dropout in the larger ones were employed to mitigate overfitting issues and stabilize training, achieving a balance between the model's capacity and its generalization to the test data.

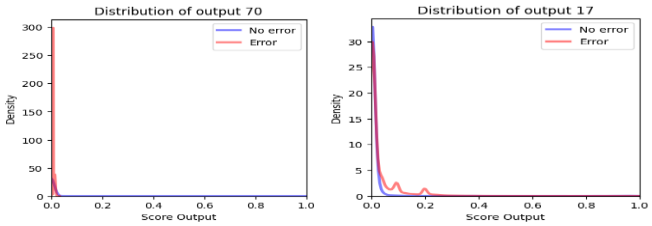


Fig. 14. Examples of distribution of image classification outputs with error-free and single errors for CLIP

B. Performance Evaluation

1) Results for BERT

During the training phase, a cross-validation with 5-folds is carried out, obtaining an accuracy of 99% (FP rate of 0.05% and FN rate of 0.1%). The final evaluation of the model is performed with 20% of the dataset (not used in the training phase), so achieving similar performance (accuracy of 99%, FP rate of 0.05% and FN of 0.1%), and furthermore demonstrating that the model generalizes well, and it is valid for detecting errors in BERT. Therefore, in the case of BERT, there is no need to check consecutive predictions to trigger the reconfiguration of the FPGA, as a FP rate of 0.05% implies an unnecessary FPGA reconfiguration for every 2,000 runs of the model. This maintains a very low FN rate (0.1%), making it very improbable to have FN (two concurrent FN every 10,000 runs and three FN every 100,000 runs). Finally, the resulting concurrent model occupies 2.48 MB, which is more than 175 times smaller than the BERT model (438 MB).

2) Results for CLIP

The resulting model is evaluated with 20% of the original dataset, achieving an accuracy of 98.6% and FP/FN rates of 11.5%/1.2%. However, an 11.5% FP rate implies an FPGA reconfiguration per approximately every 8.7 runs of the model, this is rather excessive. To reduce the number of FPGA reconfigurations, it can be performed only after N consecutive errors, for example with N_p = 3 then the average number of runs for an unnecessary reconfiguration is reduced to 742 (Fig. 12), and the mean predictions to detect a real failure with the 1.2% of FN is 3.07. The value of N_p can be adjusted to achieve the desired reconfiguration rate. The study of the best selection for the value of N and its implications for error detection is left for future work. The resulting model requires a memory of 5.68 MB, which is about 106 times less than CLIP (605 MB).

The successful application of CCED for BERT and CLIP shows that this approach is applicable for a wide range of fault models for LLMs, and different types of accelerators based on FPGAs or GPUs.

3) Concurrent classifiers implementation and robustness

The concurrent classifiers can be implemented also in hardware of FPGAs or given their low complexity, in software

running on an FPGA soft processor or even in a processor external to the FPGA as processing each input item in BERT or CLIP takes a significant amount of time. The amount of data that must be sent to the concurrent classifier is rather small, six values for BERT and 100 for CLIP for each input item. As there are several implementation options, each with different trade-offs and again given that the concurrent classifier complexity is very small compared to the main models (BERT or CLIP), the implementation of the concurrent classifiers on FPGA is left for future work.

A final consideration about the concurrent classifier is that errors can also affect it and eventually compromise error detection for the main models. In this regard, the use of Random Forest classifiers is of interest as they have been shown to be intrinsically resilient to errors due to the use of many independent trees and they can also be protected efficiently as shown in [69]. For neural networks, given the low complexity of the classifiers, they could be duplicated to ensure error detection at low overhead. Therefore, errors on the concurrent classifiers can be efficiently managed and again given the low complexity of the classifiers, they could even be duplicated to ensure error detection at low overhead. Therefore, as with the implementation, error tolerance on the classifier is not expected to be an issue and its detailed analysis is left for future work.

C. Fault location and reconfiguration for specific VPU

Reconfiguring the whole FPGA is a time-consuming process, especially for big FPGAs like the Alveo U280. In this paper, we propose to locate the faulty VPU when a fault is detected by the classifier, so that we can reconfigure only the faulty VPU within a very short time and at low overhead. The fault location scheme is based on three features of FlightLLM for common tasks.

- The MPE is reused for MMMs in all layers, so the fault on a VPU in an MPU must affect the result of the first MMM in the first layer.
- As shown in Fig. 6, different MPUs are used for the multiplication of the fixed rows of the input activation with the weight matrix, and different VPUs in an MPU work for fixed items in a row of the output matrix.
- There are always known tokens at deterministic positions, e.g. the [CLS] at the beginning in the BERT based emotion classification and the CLIP based image classification, or [BOS] and [EOS] for the beginning of a sentence and the end of a sentence.

Therefore, the location of the faulty VPU can be determined as follows (taking [CLS] as an example):

1) Perform the first MMM in the first layer offline, e.g. X*Wk, and store the first M results of the first row, denoted as y_i, $i = 1, 2, \ldots, M$, where M is the number of VPUs in an MPU (see Section IV.A);

2) Perform online the calculation of X*Wk, and compare the first M results of the first row of the output matrix y'_i with y_i ($i = 1, 2, \ldots, M$);

3) If all results match, there is no fault in MPU0; if the i-th item mismatches, we determine that the i-th VPU is faulty.

4) Reorder the MPUs (can be realized easily with an instruction for FlightLLM), so that another MPU becomes MPU0 and can be checked based on the above steps 1) to 3).

We evaluate the performance of this scheme in the PyTorch based fault injection simulator for SA0 or SA1 on each of the VPUs. Since we have 32 VPUs in an MPU, we only store the first 32 values in the first row of the result X*Wk of the first layer. The simulation results show that this scheme can locate the faulty VPU with 100% accuracy for both BERT and CLIP. This is expected because the hidden dimension is 768, and each VPU involves 32 multipliers; in this case each VPU is used 24 times for the accumulation of one item in the output matrix. It is almost impossible to have all 24 results with a specific bit always being 0 or 1. In addition, our test shows that the partial reconfiguration speed is approximately 89ms per MB on Alveo U280 using DFX in the XRT shell. Since the size of the bitstream file for a VPU is around 0.42MB, we can then estimate the reconfiguration time for a VPU to be approximately 37ms.

D. Complexity Evaluation

To evaluate the complexity of the proposed schemes, we compare the running time for normal inference, error detection and fault location for BERT and CLIP. Since the error detection and fault location schemes have not been implemented on FPGA, the comparison is based on the tests in a GPU server. The hardware and software configurations for the test are listed in Table 1, and the test results are listed in Table 2. For BERT, the times are averaged over classifications of 2000 sentences. For CLIP, the times are averaged over classifications of 2000 images. The running times of error detection and fault location are approximately 3% and 0.6% of normal inference for BERT, and about 1.1% and 0.2% for CLIP. Extrapolating on the assumption that the impact on speed of an FPGA implementation for inference and error detection would be similar, then we can argue that the complexity of error detection and fault location is negligible relative to the inference time of the original model.

Table 1. Hardware and software information for the evaluation platform

Hardware	Configuration	Software	Version
CPU	13th Gen Intel® Core™ i5-13400F×16	OS	Ubuntu 20.04
RAM	16GiB	Pytorch	1.13.0
GPU	NVIDIA GeForce RTX 4080	Numpy	1.24.3
		CUDA	11.7

Table 2. Comparison of Running Times

	BERT	CLIP
Inference	9.85s	32.49s
Error detection	0.298s	0.364
Fault location	$5.6*10^{-3}$s	$5.6*10^{-3}$s

VI. Conclusion

In this paper, the robustness of FPGA-based LLM accelerators has been studied when affected by soft errors on the configuration memory bits. To assess the effects of the errors caused by these faults (as in many cases equivalent to stuck-at faults on the VPU outputs) a novel PyTorch-based

simulator has been presented for fault injection. Two case studies (BERT-based sentence emotion classification and the CLIP-based image classification) have been evaluated. The evaluation results show that stuck-at errors on some bits of the VPU outputs can lead to a significant degradation of the model performance, and the faults on the first MPU introduce much severe performance loss than those on other MPUs. To address this issue, the use of a concurrent classifier to detect errors is proposed and evaluated. Simulation results have shown that a simple Balanced Random Forest (neural network) classifier achieves a high single error detection accuracy of 99% (98.6%) for BERT (CLIP) at a nearly negligible overhead in hardware and latency. To further increase accuracy and avoid reconfiguring the FPGA unnecessarily (e.g. for the CLIP-based image classification in this study), we have proposed performing multiple detections for successive outputs. In addition, an efficient fault location scheme has been proposed, so that the fault can be recovered by a dynamic partial reconfiguration at very low overhead.

The schemes proposed in this paper are expected to be generally applicable. So, the PyTorch based fault injection simulator is based on the hook mechanism, which can also be used for fault injection for other modules in LLMs, including activation and normalization. Moreover, the robustness analysis for different MPUs and VPUs exploits a parallel architecture, which can be used for other FPGA or GPU accelerators. Also, the design principles of the employed CCED can be used for error detection for other large models, including multi-modal and diffusion models for image and video. Finally, as the fault localization method is based on the parallel architecture of the accelerator hardware and the features of the task, then it can be easily extended to other accelerators and tasks.

REFERENCES

[1] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han, and Y. Tang, “A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development,” *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 5, pp. 1122–1136, 2023.

[2] A. Dubey, et al. The llama 3 herd of models. *arXiv preprint* arXiv:2407.21783, 2024.

[3] Gemini Team, “Gemini: A Family of Highly Capable Multimodal Models”, *ArXiv e-prints*, 2023. doi:10.48550/arXiv.2312.11805.

[4] OpenAI, “GPT-4 Technical Report”, *ArXiv e-prints*, 2023. doi:10.48550/arXiv.2303.08774.

[5] A. Q. Jiang et al, “Mistral 7B”, *ArXiv e-prints*, 2023. doi:10.48550/arXiv.2310.06825.

[6] J. Conde, M. González, P. Reviriego, Z. Gao, S. Liu and F. Lombardi, "Speed and Conversational Large Language Models: Not All Is About Tokens per Second," in Computer, vol. 57, no. 8, pp. 74-80, Aug. 2024.

[7] B. J. Kang, et al. “A survey of FPGA and ASIC designs for transformer inference acceleration and optimization” *Journal of Systems Architecture*, 2024, p. 103247.

[8] S. Zeng et al. “FlightLLM: Efficient large language model inference with a complete mapping flow on FPGAs” *in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 2024. p. 223-234.

[9] M. Ottavi et al., "Dependable Multicore Architectures at Nanoscale: The View from Europe," in *IEEE Design & Test*, vol. 32, no. 2, pp. 17-28, April 2015.

[10] M. Rabe, S. Milz and P. Mäder, "Development Methodologies for Safety Critical Machine Learning Applications in the Automotive Domain: A Survey," *in IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 129-141.

[11] I. Brett, and S. Sarkar. "LLMs for Multi-Modal Knowledge Extraction and Analysis in Intelligence/Safety-Critical Applications." arXiv preprint arXiv:2312.03088 (2023).

[12]L. Xue, L. Jie, Z. Peipei and X. Tao, "MilChat: A Large Language Model and Application for Military Equipment," 2024 7th International Conference on Machine Learning and Natural Language Processing (MLNLP), Chengdu, China, 2024.

[13] F. Su, C. Liu and H. -G. Stratigopoulos, "Testability and Dependability of AI Hardware: Survey, Trends, Challenges, and Perspectives," *in IEEE Design & Test*, vol. 40, no. 2, pp. 8-58, April 2023.

[14]A. Dorostkar, H. Farbeh and H. R. Zarandi, “An Empirical Fault Vulnerability Exploration of ReRAM-Based Process-in-Memory CNN Accelerators,” *in IEEE Transactions on Reliability*, Early Access, June 2024.

[15] Y. P. Chen et al., "Single-event Evaluation of Xilinx 16nm UltraScale+™ High-Bandwidth Memory Enabled FPGA," *IEEE Radiation Effects Data Workshop*, 2019, pp. 1-5.

[16] Z. Gao, J. Wang, R. Su, et al., "On the Dependable Operation of Bidirectional Encoder Representations from Transformers (BERT) in the Presence of Soft Errors," *in IEEE 23rd International Conference on Nanotechnology (NANO)*, 2023, pp. 582-586.

[17] C. Bernardeschi, L. Cassano, A. Domenici and L. Sterpone, "ASSESS: A Simulator of Soft Errors in the Configuration Memory of SRAM-Based FPGAs," *in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 9, pp. 1342-1355, Sept. 2014.

[18] Z. Gao et al., "Systematic Reliability Evaluation of FPGA Implemented CNN Accelerators," *in IEEE Transactions on Device and Materials Reliability*, vol. 23, no. 1, pp. 116-126, March 2023.

[19] G. Li et al., “Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications,” *in Int. Conf. High Perfor. Comput., Networking, Storage Analysis*, 2017.

[20] P. Reviriego et al., "Concurrent Classifier Error Detection (CCED) in Large Scale Machine Learning Systems," *in IEEE Transactions on Reliability*, vol. 73, no. 4, pp. 1782-1791, Dec. 2024.

[21]H. Fan, T. Chau, S. I. Venieris, et al., Adaptable Butterfly Accelerator for Attention-based NNs via Hardware and Algorithm Co-design. In 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). 599–615.

[22]T. J. Ham, Y. Lee, S. H. Seo, et al., ELSA: Hardware-Software Co-design for Efficient, Lightweight Self-Attention Mechanism in Neural Networks. In 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). 692–705.

[23]S. Hong, S. Moon, J Kim, et al., DFX: A Low-latency Multi-FPGA Appliance for Accelerating Transformer-based Text Generation. In 2022 IEEE Hot Chips 34 Symposium (HCS). 1–17.

[24]S. C. Kao, S. Subramanian, G. Agrawal, et al., FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2023). New York, NY, USA, 295–310.

[25]B. Li, S. Pandey, H. Fang, et al., FTRANS: Energy-Efficient Acceleration of Transformers using FPGA. arXiv:2007.08563 [cs.DC]

[26]L. Lu, Y. Jin, H. Bi, et al., Sanger: A Co-Design Framework for Enabling Sparse Attention Using Reconfigurable Architecture. 54th Annual IEEE/ACM International Symposium on Microarchitecture (Virtual Event, Greece) (MICRO ’21), 977–991.

[27] Y. Qin, Y. Wang, D. Deng, et al., FACT: FFN-Attention Co-Optimized Transformer Architecture with Eager Correlation Prediction. In Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA ’23), Orlando, FL, USA.

[28]H. Wang, Z. Zhang and S. Han, Spatten: Efficient sparse attention architecture with cascade token and head pruning. In 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA 2021), 97–110.

[29]H. Wang, H. Xu, Y. Wang, et al., CTA: Hardware-Software Co-design for Compressed Token Attention Mechanism. In 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA 2023), 429–441.

[30]E. Ozen and A. Orailoglu, "Sanity-check: Boosting the reliability of safety-critical deep neural network applications," 2019 IEEE 28th Asian Test Symposium (ATS), 2019, pp. 7–75.

[31]B. Reagen, U. Gupta, L. Pentecost, et al., "Ares: A framework for quantifying the resilience of deep neural networks," IEEE 55th ACM/ESDA/IEEE Design Automation Conference (DAC), 2018.

[32]Z. Gao, Y. Yao, X. Wei, et al., Reliability Evaluation of FPGA based Pruned Neural Networks, Microelectronics Reliability, vol. 130, March 2022.
[33]Xilinx, Vivado Design Suite User Guide - Dynamic Function eXchange UG909 (v2021.2) February 25, 2022.
[34]F. Libano, B. Wilson, J. Anderson, et al., "Selective Hardening for Neural Networks in FPGAs," IEEE Trans. Nuclear Science, vol. 66, no. 1, pp. 216-222, 2019.
[35]C. Israel, B. Fabio, L.K. Fernanda, A.A. Susin, P. Rech, "Reliability Analysis on Case-Study Traffic Sign Convolutional Neural Network on APSoC," in Proc. IEEE 19th LATS, pp. 1-6, 2018.
[36]F. Libano, P. Rech, B. Neuman, et al., "How Reduced Data Precision and Degree of Parallelism Impact the Reliability of Convolutional Neural Networks on FPGAs," IEEE Trans. Nuclear Science, vol. 68, no. 5, pp. 856-872, Jan. 2021.
[37]D. Xu, Z. Zhu, C. Liu, et al., "Reliability Evaluation and Analysis of FPGA-Based Neural Network Acceleration System," IEEE Trans. VLSI Systems, vol. 29, no. 3, Jan. 2021.
[38]Z. Gao, S. Gao, Y. Yao, et al., Systematic Reliability Evaluation of FPGA Implemented CNN Accelerators, IEEE Trans. Device and Materials Reliability, vol. 23, no. 1, March 2023.
[39]Z. Gao, Jiaqi Feng, S. Gao, et al., Modeling the Effect of SEUs on the Configuration Memory of SRAM-FPGA based CNN Accelerators, IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 14, no. 4, pp. 799-810, Dec. 2024.
[40]Z. Gao, J. Deng, P. Reviriego, et al., Reducing the Energy Dissipation of Large Language Models (LLMs) with Approximate Memories, in Proc IEEE Int. Symp. Circuits Syst., Singapore, May 19-22, 2024.
[41]Z. Gao, L. Yuan, P. Reviriego, et al., Dependability Evaluation of Stable Diffusion with Soft Errors on the Model Parameters, IEEE 24th International Conference on Nanotechnology (NANO 2024), Gijon, Spain, Jul. 2024.
[42]Z. Gao, S. Liu, P. Reviriego, et al., Dependability and Protection of Transformer Models against Soft Errors on Text Embeddings, *IEEE Transactions on Device and Materials Reliability*, Early Access, Oct. 2024.
[43]Z. Gao, J. Deng, Reviriego, et al., Operating Conversational Large Language Models (LLMs) in the Presence of Errors: The Case of Mistral-7B, IEEE Nanotechnology Magazine, Early Access, Dec. 2024.
[44]F. F. dos Santos, L. Draghetti, L. Weigel, et al., "Evaluation and mitigation of soft-errors in neural network based object detection in three gpu architectures," 47th Annual IEEE/IFIP DSN-W, 2017.
[45]Z. Xu and J. Abraham, "Safety design of a convolutional neural network accelerator with error localization and correction," 2019 IEEE International Test Conference (ITC), pp. 1–10, 2019.
[46]K. Zhao, S. Di, S. Li, et al., "FT-CNN Algorithm-Based Fault Tolerance for Convolutional Neural Networks," IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 7, pp. 1677-1689, July 2021.
[47]T. Marty, T. Yuki and S. Derrien, "Safe Overclocking for CNN Accelerators Through Algorithm-Level Error Detection," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 12, pp. 4777-4790, Dec. 2020.
[48]Z. Gao, Y. Qi, J. Shi, et al., Detect and Replace: Efficient Soft Error Protection of FPGA based CNN Accelerators, IEEE Transactions on Very Large Scale Integration (VLSI), vol. 33, no. 1, pp. 66-74, Jan. 2025.
[49]Z. Gao, H. Zhang, Y. Yao, et al. Soft error tolerant convolutional neural networks on FPGAs with ensemble learning[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2022, 30(3): 291-302.
[50]Z. Gao, H. Zhang, X. Wei, et al. Ensemble of Pruned Networks for Reliable Classifiers[C]//2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2021.
[51]I. Herrera-Alzu and M. Lopez-Vallejo, "Design Techniques for Xilinx Virtex FPGA Configuration Memory Scrubbers," IEEE Transactions on Nuclear Science, vol. 60, no. 1, pp. 376-385, Feb. 2013.
[52]R. Zhang, L. Xiao, J. Li, X. Cao and L. Li, "An Adjustable and Fast Error Repair Scrubbing Method Based on Xilinx Essential Bits Technology for SRAM-Based FPGA," IEEE Transactions on Reliability, vol. 69, no. 2, pp. 430-439, June 2020.
[53]J. Devlin, M. Chang, K. Lee, et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," NAACL 2019, Minneapolis, USA.
[54]Code from Huggingface for sentence emotion classification, https://huggingface.co/Vasanth/bert-base-uncased-finetuned-emotion.
[55]A. Radford, J. W. Kim J W, C. Hallacy, *et al.*, "Learning transferable visual models from natural language supervision," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Virtual, Online, vol. 139, pp. 8748-8763, Jul. 2021.
[56]Code for CLIP from Github, https://github.com/openai/CLIP.
[57]I. Tuzov, D. de Andrés, J. -C. Ruiz, et al., "BAFFI: a bit-accurate fault injector for improved dependability assessment of FPGA prototypes," 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 2023, pp. 1-6,.
[58]F. Ferlini, F. Viel, L. O. Seman, et al., "A Methodology for Accelerating FPGA Fault Injection Campaign Using ICAP," *Electronics*, vol. 12, no. 4, Feb. 2023.
[59]Xilinx, Xilinx® Runtime (XRT) Architecture, https://xilinx.github.io/XRT/master/html/index.html, accessed on Dec. 30, 2024.
[60]G. Xiao, J. Lin, M. Seznec, et al., SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models, ICML 2023.
[61]C. Kuan-Hsun, C. Su, C. Hakert, et al., "Efficient realization of decision trees for real-time inference," ACM Transactions on Embedded Computing Systems 21.6 (2022): 1-26.
[62]Y. Darren and M. Z. Islam. "FastForest: Increasing random forest processing speed while maintaining accuracy," Information Sciences 557 (2021): 130-152.
[63]Z. Xue, J. Wei and W. Guo, "A Real-Time Naive Bayes Classifier Accelerator on FPGA," in *IEEE Access*, vol. 8, pp. 40755-40766, 202.
[64]P. Ginsparg. "How many coin flips on average does it take to get n consecutive heads?", 2005, https://www.cs.cornell.edu/~ginsparg/physics/INFO295/mh.pdf
[65]G. Lemaître, F. Nogueira and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," Journal of machine learning research, vol. 18, no. 17, pp. 1-5, 2017
[66]Balanced Random Forest Classifier Library, https://imbalanced-learn.org/stable/references/generated/imblearn.ensemble.BalancedRandomForestClassifier.html
[67]F. Chollet *et al*., "Keras", https://keras.io, 2015.
[68]N. V. Chawla, K. W. Bowyer, et al., SMOTE: synthetic minority over-sampling technique, Journal of artificial intelligence research, vol. 16, pp. 321-357, 2002.
[69]S. Liu, P. Reviriego, P. Montuschi et al., Error-Tolerant Computation for Voting Classifiers With Multiple Classes, *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 13718-13727, Nov. 2020.