Soft Error Resilience Analysis of LSTM Networks

Christopher P Vasquez
cvasqu6@lsu.edu
Louisiana State University
Baton Rouge, LA, USA

Travis LeCompte
tdleco95@gmail.com
Louisiana State University
Baton Rouge, LA, USA

Xu Yuan
xyuan@udel.edu
University of Delaware
Newark, DE, USA

Nian-Feng Tzeng
nianfeng.tzeng@louisiana.edu
University of Louisiana at Lafayette
Lafayette, LA, USA

Lu Peng
lpeng3@tulane.edu
Tulane University
New Orleans, LA, USA

ABSTRACT

Long Short-Term Memory (LSTM) deep neural networks are diverse in the tasks they can accomplish, such as image captioning and speech recognition. However, they remain susceptible to transient faults when deployed in environments with high-energy particles or radiation. It remains unknown how the potential transient faults will impact LSTM models. Therefore, we investigate the resilience of the weights and biases of these networks through four implementations of the original LSTM network. Based on the observations made through the fault injection of these networks, we propose an effective method of fault mitigation through Hamming encoding of selected weights and biases in a given network.

CCS CONCEPTS

• **Hardware → System-level fault tolerance**.

KEYWORDS

Deep Neural Networks, Fault Tolerance, LSTM

ACM Reference Format:
Christopher P Vasquez, Travis LeCompte, Xu Yuan, Nian-Feng Tzeng, and Lu Peng. 2024. Soft Error Resilience Analysis of LSTM Networks. In *Great Lakes Symposium on VLSI 2024 (GLSVLSI '24), June 12–14, 2024, Clearwater, FL, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3649476.3658776

1 INTRODUCTION

The recent revolution in Deep Neural Networks (DNNs) has significantly changed many areas, including finance, manufacturing, image processing, natural language processing, etc. Typical DNNs include Convolutional Neural Networks (CNNs), LSTMs, and recently-developed Transformers. Like most DNNs, LSTMs are often deployed on CPUs, GPUs, and FPGAs. This computational hardware allows DNNs to perform the needed computations and allows various DNN architectures to be deployed. However, given that DNNs require this hardware to function, environmental factors can affect the hardware, affecting the neural network. Since LSTMs can be deployed in applications that can be used in a wide variety of environments, these networks can encounter various environmental susceptibilities. In particular, environments with the presence of cosmic particles or radiation can cause transient faults in DNNs [2]. This type of fault occurs when a bit in a storage element becomes flipped to another state, leading to temporary soft errors, known as Single Event Upsets (SEUs) [11]. Many studies have been conducted to research and improve the resilience of hardware and software due to these anomalies [1, 3, 5].

One of the most common ways to model SEUs and other transient faults is through fault injection of these DNNs. Specifically, this process involves randomly selecting elements within a neural network to perform bit flips upon. For this study, we utilize TensorFI2 [2] for fault injection, as this injector allows for flexible parameters for customizing injections for TensorFlow-based models. Due to limitations of the current version of TensorFI2, it has difficulty working with newly-developed Transformer models with much larger model sizes than LSTM models. On the other hand, both LSTM and Transformer work for sequence-to-sequence tasks and can handle long-range dependencies with non-linear transformations. Therefore, we investigate LSTM in this paper while soft error resilience of CNNs has been well studied [4, 7, 10].

In this work, we developed four different binary sentiment analysis models with various architectures and performed fault injections on each to measure their resilience. By introducing multiple numbers of errors under other conditions, we were able to determine areas of susceptibility that lead to decreases in each model's accuracy concerning the weights and biases. Based on these findings, we developed a means of mitigating the errors by encoding the most susceptible bits of the weights and biases through the Hamming Code, as this method only requires additional computation. To reduce computation costs, we only selected specific sets of these weights and biases to encode based on their susceptibility. Implementing our method into each network, we compared and proved the effectiveness of the Hamming Code as a method of fault mitigation, significantly improving the resilience of LSTM networks.

In summary, the contributions of this paper can be listed as follows:
- We employed a recent fault injector to analyze the vulnerabilities of LSTM models.
- We developed a method to improve the resilience of LSTMs via Hamming code.
- Based on the experiments, the proposed error mitigation method

GLSVLSI '24, June 12–14, 2024, Clearwater, FL, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0605-9/24/06
https://doi.org/10.1145/3649476.3658776

can significantly improve the resilience of LSTMs even when 20 faults were injected.

2 LSTMS FOR FAULT INJECTION & MITIGATION

To perform the work in this study, we constructed four different LSTM networks with different architectures for binary sentiment analysis. Given the purpose of these neural networks, they all use an embedding layer as the input layer, allowing for the text passed to the network to be interpretable. This layer only consists of a single matrix of weights. For all remaining layers, save for the last layer, the architecture of each network varies. In order to cover different variations, we decided to use a single LSTM, stacked LSTMs, a bidirectional LSTM, and stacked bidirectional LSTMs for each network, respectively.

The dataset used for training and testing with the LSTM neural networks is Stanford's Large Movie Review dataset [8]. This dataset provides 50,000 highly polar movie reviews, with 25000 for training and 25000 for testing. Given that this dataset is intended for binary sentiment classification of movie reviews, it consists of two different classes: "positive" and "negative." This dataset can be freely obtained online and is commonly used due to its large volume of data when compared to other benchmark datasets. The LSTM models were trained with the Adam optimizer and had a learning rate of 0.001 and a batch size of 128.

3 FAULT INJECTION AND PROTECTION

3.1 Fault Model & Injection

Before implementing any type of mitigation into LSTM networks, we needed to identify areas of susceptibility within the four different networks, as seen in phase 0 of Figure 1. Therefore, for our fault model, we assume that the training process is fault-free. We make this assumption as most neural networks will be trained in a stable environment before being deployed in a potentially susceptible area. The lack of such an assumption would require additional analysis and testing of potential bit flips during the training phase. For randomly selected weights and biases from each layer, we perform random bit-flips and observe the network's final output. This process is based on other fault injection methods used in prior work within this area [2, 6, 9].

An important aspect to note is that we consider only SEUs that would not lead to obvious failures, such as a system crash. Therefore, we will only consider faults that result in silent data corruption (SDC), which is not easy to detect by the users. Furthermore, this fault model simulates the SEUs or bit-flip faults, which can occur in data stored within the memory of a system.

As previously mentioned, the data to be flipped consists of the weights' and biases' bits in the networks. An example of this process can be seen in Figure 2, where the fourth most significant bit of a given weight is flipped. An important aspect of these weights is that they are in FP32 format. Specifically, the figure depicts the original value increasing by a factor of approximately 4 billion, showing the drastic changes that can result from a single bit-flip.

With this fault model in mind, we individually inject each set of weights and biases in each layer for each of the 4 LSTM networks. Furthermore, the weights and biases from each set are randomly chosen. For our study, we chose to perform 100 fault injections for each set of weights and biases in each layer. We repeated this process for 1, 10, and 20 fault injections. Once the injections were performed, we measured the accuracy of the injected model based on the 25000 sample test set. Given our weight-set approach for injection, we are able to identify both sets of weights/biases and layers of the network that are most susceptible to SEUs. Furthermore, variation in the number of faults allows us to stress-test the resilience of the network, further highlighting susceptible weight/bias sets and layers.

To continue this investigation, we also looked into the effects of injecting specific sections of the FP32 format. We chose to limit the bit-field to be injected into (i.e., the sign, exponent, and mantissa) for individual experiments while maintaining the previous parameters. This analysis will allow us to have more insight into the bits that need to be hardened within specific weight sets and layers.

3.2 Protection with Hamming Code

Based on the observations of our fault injection of the four networks, we developed a method to improve the resilience of LSTMs via Hamming code. Therefore, we first created Hamming encoding and decoding functions specifically designed for the 8-bit exponent of FP32 numbers to implement (7,4) Hamming code on the two 4-bit halves of the exponent of the LSTM models. We chose only to encode this section of bits, as it is the most susceptible to SEUs. Additionally, our decision to split the exponent bits into two 4-bit halves allows us to detect two errors (assuming these errors do not occur within the same half).

Since the addition of these functions to the model invokes extra computation, it is important to harden only the exponent bit field of weights and biases that are susceptible to SEUs. For instance, we noticed that the embedding layer, in our case, was inherently resilient, even with exponent bit field SEUs. Therefore, implementing Hamming code on these weights would introduce unnecessary computation, emphasizing the importance of conducting phase 0 as opposed to hardening the entire model. In our case, since the initial accuracy of every model was approximately 86.5%, we selected sets of weights and biases that produced an average accuracy of less than 80% when injected with 20 SEUs in the exponent bit field (See Figure 2). We selected this specific testing case as it represented the worst-case scenario.

Since the models' weights and biases will become encoded, we must ensure that hamming decoders are in the layers that contain susceptible weights and biases to correct any faults. These decoders allow the weights and biases to be corrected if bit one error is present. Furthermore, these modules restore the original values of the weights and biases, allowing them to be utilized by the models. As seen in phase 1 of Figure 1, we modified the selected layers for hardening and wrapped instances of memory accesses to weights and biases with the "K.in_train_phase()" function. This specific function allows layers to behave differently during training and inferencing. This behavior is important to ensure that our method works for both untrained and trained models. Its functionality is similar to an if-else statement where the condition is determined by the model's deployment state. For the training state, we had the model utilize unencoded weights, allowing it to develop trained

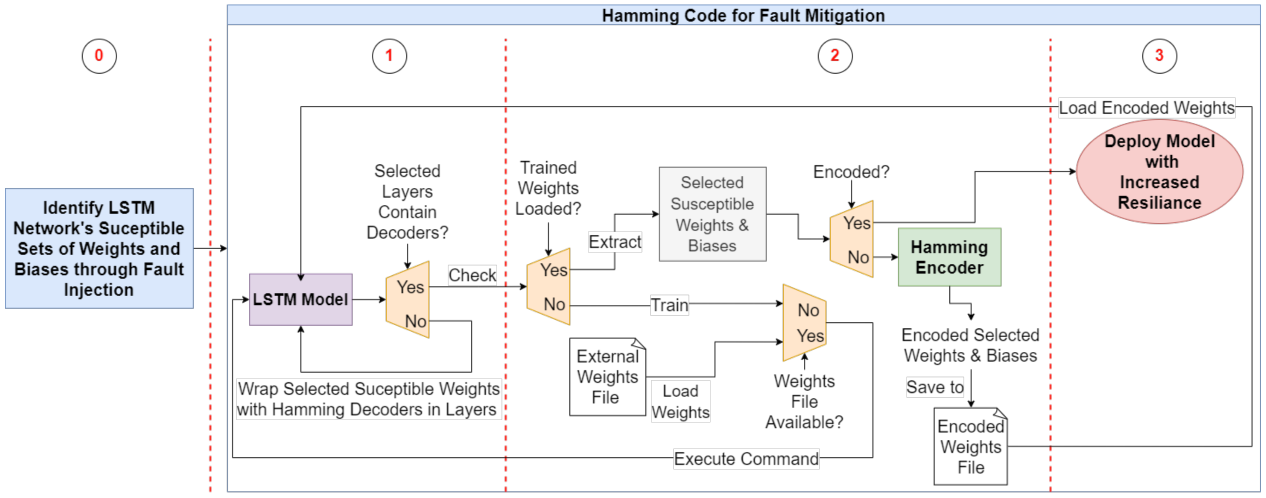

Hamming Code for Fault Mitigation
0
1
2
3
Identify LSTM Network's Suceptible Sets of Weights and Biases through Fault Injection
LSTM Model
Selected Layers Contain Decoders?
Yes
No
Check
Wrap Selected Suceptible Weights with Hamming Decoders in Layers
Trained Weights Loaded?
Yes
No
Extract
Train
Selected Susceptible Weights & Biases
External Weights File
Load Weights
Weights File Available?
No
Yes
Execute Command
Encoded?
Yes
No
Hamming Encoder
Encoded Selected Weights & Biases
Save to
Encoded Weights File
Load Encoded Weights
Deploy Model with Increased Resiliance

Figure 1: System Overview of Implementing Hamming Code for Fault Mitigation of LSTM Networks

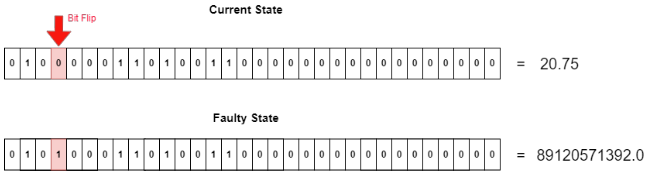


Figure 2: Bit-Flip on Weight in Single-Precision Floating Point Format

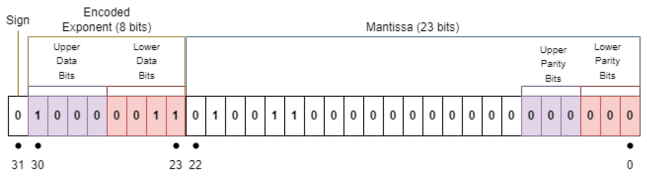


Figure 3: Arrangement of Data and Parity Bits for Hamming Encoding

values for prediction. Assuming that we encode the weights after this training, the weights passed to the network will not be set to their trained values. Therefore, for the inference state, we have the model call our decoding function (hamming_decode_weights) on the passed encoded weights to correct any errors.

Once the decoders are in place, we can encode the weights of the networks. If the model is pre-trained, we simply load the original weights from a saved weights file into the model and encode them with the previously mentioned encoder function–which applies a (7,4) Hamming encoding. If the model has not been trained yet, we can simply train the model to obtain the necessary trained weights and perform the encoding immediately after. Regardless of the situation, we will be able to save the encoded weights to a weights file that can be loaded at any time by the LSTM model. This process can be seen more clearly in phase 2 of Figure 1.

Table 1: Accuracies of the Original and Modified LSTM Networks

Model	Original Accuracy (%)	New Accuracy (%)
LSTM	87.09999918937683	87.09999918937683
Stacked LSTM	86.72800064086914	86.72800064086914
Bidirectional LSTM	86.88799738883972	86.88799738883972
Stacked Bidirectional LSTM	87.81599998474121	87.81200051307678

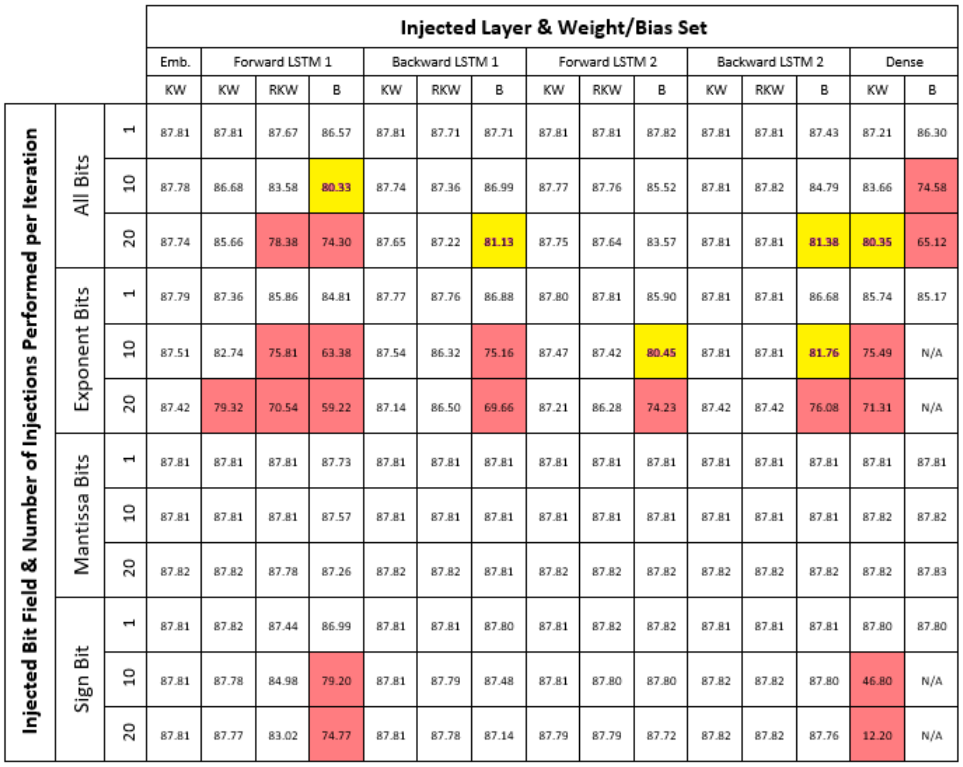
As previously mentioned, we chose to encode the exponent bit field of susceptible weights and biases, as SEUs in this section of bits lead to SDC consistently. Since we are encoding two separate sets of 4-bit data with (7, 4) Hamming code, we will have a total of 6 parity bits. In other words, we will have an additional 6 bits for every encoded exponent bit field of a weight. Given that we are using FP32 numbers, we cannot expand the number of bits of these data types. Doing so would require us to use FP64, which would potentially lead to even more issues, as a wider range of values and bits would be available for SDC. Since we already proved that the mantissa was the least affected by changes in the bits, we chose to store each set of 3 parity bits in the least significant bits of each FP32 number, as seen in Figure 3. This systematic arrangement of bits for encoding is handled by the encoder function. Additionally, as shown in Table 1, this change in the least significant bits of weights and biases of the selected sets had little to no change in each model's accuracy.

With both the encoded weights/biases and the decoders, our network can simply load the weights from the previously created encoded weights file and deploy the model with increased resilience, as seen in phase 3 of Figure 1. All matrix operations for this implementation were done through Keras and Tensorflow APIs to ensure that any Keras/Tensorflow-based model could utilize our mitigation methods without potential dependency issues. Additionally, this implementation does not require any retraining of the models.

4 RESULTS & ANALYSIS

We performed the experiments on a workstation consisting of an AMD Ryzen 5950X 16-core CPU, 64GBs of 3600MHz RAM, NVIDIA

Table 2: Matrix of the Average Accuracies of Stacked Bidirectional LSTM's Susceptible Weight/Bias Sets After 1, 10, and 20 SEUs Over 100 Iterations Without Protections (Red cells have accuracies below the 80% threshold and yellow cells are closing to the threshold.)



Injected Bit Field & Number of Injections Performed per Iteration			Injected Layer & Weight/Bias Set														
			Emb.	Forward LSTM 1			Backward LSTM 1			Forward LSTM 2			Backward LSTM 2			Dense	
			KW	KW	RKW	B	KW	RKW	B	KW	RKW	B	KW	RKW	B	KW	B
	All Bits	1	87.81	87.81	87.67	86.57	87.81	87.71	87.71	87.81	87.81	87.82	87.81	87.81	87.43	87.21	86.30
		10	87.78	86.68	83.58	80.33	87.74	87.36	86.99	87.77	87.76	85.52	87.81	87.82	84.79	83.66	74.58
		20	87.74	85.66	78.38	74.30	87.65	87.22	81.13	87.75	87.64	83.57	87.81	87.81	81.38	80.35	65.12
	Exponent Bits	1	87.79	87.36	85.86	84.81	87.77	87.76	86.88	87.80	87.81	85.90	87.81	87.81	86.68	85.74	85.17
		10	87.51	82.74	75.81	63.38	87.54	86.32	75.16	87.47	87.42	80.45	87.81	87.81	81.76	75.49	N/A
		20	87.42	79.32	70.54	59.22	87.14	86.50	69.66	87.21	86.28	74.23	87.42	87.42	76.08	71.31	N/A
	Mantissa Bits	1	87.81	87.81	87.81	87.73	87.81	87.81	87.81	87.81	87.81	87.81	87.81	87.81	87.81	87.81	87.81
		10	87.81	87.81	87.81	87.57	87.81	87.81	87.81	87.81	87.81	87.81	87.81	87.81	87.81	87.82	87.82
		20	87.82	87.82	87.78	87.26	87.82	87.82	87.81	87.82	87.82	87.82	87.82	87.82	87.82	87.82	87.83
	Sign Bit	1	87.81	87.82	87.44	86.99	87.81	87.81	87.80	87.81	87.82	87.82	87.81	87.81	87.81	87.80	87.80
		10	87.81	87.78	84.98	79.20	87.81	87.79	87.48	87.81	87.80	87.80	87.82	87.82	87.80	46.80	N/A
		20	87.81	87.77	83.02	74.77	87.81	87.78	87.14	87.79	87.79	87.72	87.82	87.82	87.76	12.20	N/A

RTX 3090 GPU, and the Windows 10 operating system. Additionally, Tensorflow 2.11.0 was utilized in conjunction with Keras 2.11.0 for developing the LSTM models. Although we performed tests on four different architectures, for this work, we will mainly focus on the most complex model–i.e., the stacked bidirectional LSTM model–as all models consist of very similar results.

4.1 LSTM Models' Resilience

Table 2 depicts the average accuracies for all the sets of weights and biases of the stacked bidirectional LSTM which underwent fault injection without protections. Based on the observations made from the SEU injection experiments, we have noticed that the large number of weights within the embedding layer makes it inherently resilient to our SEU tests. However, the subsequent layer of the embedding layer tends to be most susceptible to faults within all models. Furthermore, biases generally undergo SDC when SEUs are introduced in the exponent bit field, especially when multiple are introduced. Lastly, since the dense layer contains a small number of weights and acts as the output layer, this layer is extremely sensitive to SEUs and is almost always susceptible to SEUs. More specifically, the dense layer seems only to be majorly affected when multiple faults are introduced in the exponent bit field or when multiple weights or biases experience SEUs in their sign bits, as seen in Table 2. Given that we are performing binary classification, the sensitivity of the sign bits is to be expected, as the sign of the output determines the prediction outcome.

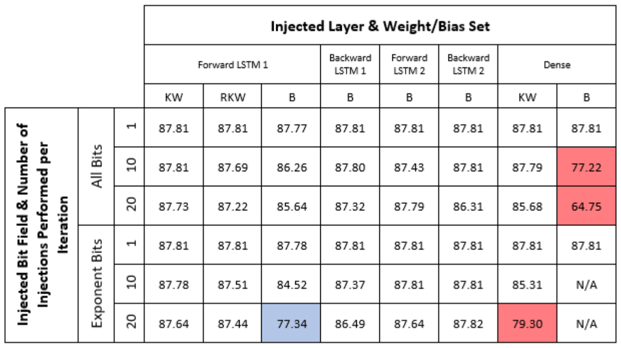
4.2 Mitigation Performance

To maintain consistency with testing, we continued to inject SEUs in sets of 1, 10, and 20 for 100 iterations for each layer. However, since we hardened specific sets of weights and biases within the network, we performed injection tests only on the sets with our mitigation method implemented. Furthermore, given that we only hardened the exponent bit field of the weights and biases, we performed injection into only the exponent for our bit field tests. In addition to this set of injections, we also continued to inject all the bits to verify that layers that have our decoding method do not affect the overall performance of the model even when they are not the ones experiencing errors.

Table 3 depicts the average accuracies for the susceptible sets of weights and biases that we applied our mitigation method on for the stacked bidirectional LSTM (Please note that red signifies accuracies below the 80% threshold while blue signifies an accuracy below the threshold but showed significant improvement in accuracy). We only show the susceptible weights and biases we chose to inject, as the hardening of other layers did not show significant signs of each model's performance against bitflips.

In Table 3, the dense layer's bias continues to be a weak point due to the minimum number of biases present in the layer. Given

Table 3: Matrix of the Average Accuracies of Stacked Bidirectional LSTM's Susceptible Weight/Bias Sets After 1, 10, and 20 SEUs Over 100 Iterations with Protections

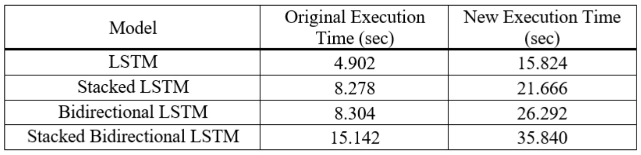


Injected Bit Field & Number of Injections Performed per Iteration			Injected Layer & Weight/Bias Set							
			Forward LSTM 1			Backward LSTM 1	Forward LSTM 2	Backward LSTM 2	Dense	
			KW	RKW	B	B	B	B	KW	B
	All Bits	1	87.81	87.81	87.77	87.81	87.81	87.81	87.81	87.81
		10	87.81	87.69	86.26	87.80	87.43	87.81	87.79	77.22
		20	87.73	87.22	85.64	87.32	87.79	86.31	85.68	64.75
	Exponent Bits	1	87.81	87.81	87.78	87.81	87.81	87.81	87.81	87.81
		10	87.78	87.51	84.52	87.37	87.81	87.81	85.31	N/A
		20	87.64	87.44	77.34	86.49	87.64	87.82	79.30	N/A

the low number of biases–i.e., one bias–multiple injections within the same values lead to this detrimental decrease in accuracy. This same behavior can be seen in the kernel weights of the network, as 20 SEUs in the exponent bit field lead to the most significant decrease in accuracy. However, when compared to the original network's results, our modified model maintains a better accuracy. The remaining layers of the network all showed significant signs of improvement, all being around the maximum attainable accuracy. The one exception to this behavior is when 20 SEUs were introduced into the first forward LSTM layer. Although this accuracy of 77.34% is below our threshold, we still regard it as an improvement, as the accuracy for this test case with the original model was significantly worse at 59.22% as shown in Table 2. Note that there is a slight accuracy drop in the case of all bits and dense bias with 20 SEUs. This happened due to the large number and the randomness of the injections. Based on these observations, the overall results of this model show the improved resilience of LSTM networks with our mitigation method.

4.3 Execution Time

Table 4: Average Execution Times Over 10 Iterations of Original and Modified LSTM Models on Large Movie Dataset's Test Set



Model	Original Execution Time (sec)	New Execution Time (sec)
LSTM	4.902	15.824
Stacked LSTM	8.278	21.666
Bidirectional LSTM	8.304	26.292
Stacked Bidirectional LSTM	15.142	35.840

As a final analysis of our model, we measured the execution time of the original model against our modified model. This experiment was conducted by having each model compute prediction for the Large Movie Dataset's test set (25000 samples) for ten iterations. The times for each iteration were then averaged to give us the values shown in Table 4.

Given that we introduced additional computation into the model for decoding the encoded exponent bits, we expected to see an increase in the average execution time of the model. Based on the obtained results, we can observe a clear tradeoff between execution time and model resilience. Specifically, our modified models ran anywhere from 2 to 3 times the original execution time. Therefore, when deciding to harden specific weight sets or layers, it is important to consider the use case of the model and the number of elements being hardened, as each incurs a cost.

5 CONCLUSION

By exploring the resilience of LSTM networks, we discovered that, like many other neural networks, LSTMs share similar susceptibilities to other neural networks due to the usage of FP32. Usage of this data type with neural network models leaves an excessive range for the weights and biases, leading to SEUs having major impacts on the performance of models. Additionally, we also saw that different layers and sets of weights and biases of LSTM networks can have varying susceptibilities. However, a common area of susceptibility was the exponent bit-field of FP32 numbers. Through these discoveries, we were able to use Hamming code with the weights and biases to reduce the rate of SDC within the neural networks, leading to increased accuracies under various testing conditions. We hope that future research can build upon our methodology and combine it with others to improve the overall resilience of other neural networks.

REFERENCES

[1] Sui Chen, Greg Bronevetsky, Lu Peng, Bin Li, and Xin Fu. 2016. Soft error resilience in Big Data kernels through modular analysis. *The Journal of Supercomputing* 72 (2016), 1570–1596.

[2] Zitao Chen, Niranjhana Narayanan, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2020. Tensorfi: A flexible fault injection framework for tensorflow applications. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 426–435.

[3] Lide Duan, Bin Li, and Lu Peng. 2009. Versatile prediction and fast estimation of architectural vulnerability factor from processor performance metrics. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. IEEE, 129–140.

[4] Zhen Gao, Han Zhang, Yi Yao, Jiajun Xiao, Shulin Zeng, Guangjun Ge, Yu Wang, Anees Ullah, and Pedro Reviriego. 2022. Soft error tolerant convolutional neural networks on FPGAs with ensemble learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 3 (2022), 291–302.

[5] Travis LeCompte, Walker Legrand, Sui Chen, and Lu Peng. 2017. Soft error resilience of Big Data kernels through algorithmic approaches. *The Journal of Supercomputing* 73 (2017), 4739–4772.

[6] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. 2017. 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*. 1–12.

[7] Lucas Matana Luza, Annachiara Ruospo, Daniel Söderström, Carlo Cazzaniga, Maria Kastriotou, Ernesto Sanchez, Alberto Bosio, and Luigi Dilillo. 2021. Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks. *IEEE Transactions on Emerging Topics in Computing* 10, 4 (2021), 1867–1882.

[8] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*. 142–150.

[9] Elaheh Malekzadeh, Nezam Rohbani, Zhonghai Lu, and Masoumeh Ebrahimi. 2021. The impact of faults on DNNs: A case study. In *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 1–6.

[10] Zheyu Yan, Yiyu Shi, Wang Liao, Masanori Hashimoto, Xichuan Zhou, and Cheng Zhuo. 2020. When single event upset meets deep neural networks: Observations, explorations, and remedies. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 163–168.

[11] James F Ziegler. 1996. Terrestrial cosmic rays. *IBM journal of research and development* 40, 1 (1996), 19–39.