Fault Injection in Transformer-Based Neural Networks: A Comprehensive Experimental Study

Ashwin Sudhir Sonawane*, Shreyas Khandare†, Qasim Bhabhrawala‡
*Dept. of Computer Science, Florida State University, Email: as22dk@fsu.edu
†Dept. of Computer Science, Florida State University, Email: svk23@fsu.edu
‡Dept. of Computer Science, Florida State University, Email: qb23a@fsu.edu

***Abstract*—This paper presents a detailed experimental study on the effects of hardware-like fault injection in a transformer-based neural network trained on the MNIST dataset. We compare a clean model (`clean.py`), achieving ∼94.55% test accuracy, to a fault injection model (`fault.py`) simulating various fault types (dropout, perturbation, bit-flip, bias shift) at severities 0.05, 0.10, 0.20, and 0.30. Our study documents the challenges encountered, including learning rate instabilities, partial steps (like PCA-based internal analysis) deferred for future work. We provide extensive visualizations (12 figures) showing training loss, clean accuracy, fault injection trends, and accuracy drops. Results confirm that fault severity strongly correlates with accuracy degradation, emphasizing the need for robust training strategies and improved hardware fault tolerance.**

***Index Terms*—Fault Injection, Transformer, Neural Networks, MNIST, Hardware Faults, Robustness, GPU Acceleration**

I. Introduction

Transformer architectures have increasingly become a cornerstone in the realm of modern deep learning, particularly celebrated for their potent attention mechanisms and adaptable design. This flexibility allows transformers to excel in a variety of applications, from natural language processing to image recognition. However, real-world hardware can introduce both transient and permanent faults that potentially degrade the performance of neural networks. Prompted by the critical need to investigate and comprehend the impact of these hardware faults on model reliability, we embarked on developing a transformer-based model from scratch, leveraging CuPy for GPU acceleration to enhance computational efficiency. Initially, our development began on CPU platforms, but due to the demanding nature of the training processes, we swiftly transitioned to utilizing GPU resources, thereby managing to conduct our training within a more reasonable timeframe.

This paper meticulously details our journey in constructing and scrutinizing two primary codebases:

- **`clean.py`**: This script embodies a pristine transformer classifier that impressively achieves an accuracy of 94.55% on the MNIST dataset, showcasing the model's robustness and the effectiveness of our implementation.
- **`fault.py`**: This model introduces a fault injection mechanism that simulates various types of disturbances such as dropout, perturbation, bit-flip, and bias shift at differing levels of severity. The purpose of this script is to evaluate the resilience of the transformer architecture under simulated hardware fault conditions.

Originally, our methodology included plans for conducting repeated trials to ensure statistical significance. However, upon obtaining consistent results across initial experiments, we determined that further repeated runs were unnecessary. Additionally, we contemplated integrating robust training techniques, such as noise injection during the training phase, and analyzing internal representations using methods like PCA or t-SNE. Nevertheless, these exploratory steps have been deferred to future research endeavors to further enhance the understanding and performance of transformer models under fault conditions.

II. Background and Motivation

- We implement a **transformer-based** approach, which is predominantly utilized in natural language processing tasks. In our research, we apply this method to the MNIST dataset to rigorously test its resilience to hardware-like faults, assessing its effectiveness in a new domain.
- We **systematically simulate four distinct types of faults** at multiple levels of severity. This comprehensive simulation provides a broad and detailed perspective on how various errors can influence the outputs of the network, allowing us to better understand fault impacts across different scenarios.
- We openly discuss the **challenges and partial failures** encountered in our methodology, such as the inability to successfully train the fault model with a learning rate of 0.001, which resulted in NaN values. Highlighting these issues offers insights into the complexities and limitations of applying advanced neural network architectures under fault conditions.

III. Experimental Setup

A. Model Architecture

Our experimental neural network configuration processes images of dimensions 28×28 pixels, dividing each image into patches of 7×7, resulting in a total of 16 distinct patches per image. These patches are subsequently linearly embedded into vectors of 128 dimensions ($d_{\text{model}} = 128$). The core processing is performed by multiple stacked transformer layers, each consisting of self-attention mechanisms and feedforward neural networks. Importantly, the embedding of the first patch (often referred to as the CLS token) is specifically utilized as the

input to a final linear classifier head, which performs the task of classifying the images.

B. GPU vs. CPU Decision

Initially, our training experiments were conducted using CPU resources. However, we soon discovered that CPU-based training was markedly inefficient for our needs, particularly for repeated training cycles spanning 100 epochs each. To enhance efficiency, we transitioned to utilizing GPU acceleration with the assistance of the CuPy library, which drastically reduced our training durations from several hours to approximately half an hour. This significant decrease in training time facilitated multiple experimental runs under varied fault injection scenarios, enhancing the robustness of our findings.

C. Dataset and Training (Clean Model)

For our experiments, we employed the MNIST dataset, accessed through the `mnist.npz` file. Our preprocessing steps involved normalizing the input images and converting them into a flat vector format. Using the `clean.py` script, the network:

- Underwent training for 100 epochs with a learning rate (LR) of 0.001, a batch size of 128, and an optimizer similar to Adam, all from an uninitialized state.
- Achieved a final average loss of approximately 0.0103, with a total training time of about 1872 seconds when utilizing GPU resources.
- Attained a test accuracy of **94.55%**.

Despite initial plans for conducting multiple trials to analyze variance, consistent convergence to the same accuracy across different runs led us to forego repeated trials.

D. Fault Injection Model

The `fault.py` script is designed to modify the forward pass of the network by integrating faults after each transformer layer, simulating real-world scenarios where hardware or software errors might occur. These faults include:

- **Dropout**: This involves zeroing out a random fraction of the outputs to simulate temporary loss of data transmission or processing capabilities.
- **Perturbation**: Gaussian noise is added to the outputs to mimic the effect of electrical noise or minor hardware imperfections.
- **Bit-Flip**: This simulates a more severe form of hardware error where the signs of the outputs are flipped with a predefined probability.
- **Bias Shift**: Outputs are scaled by a factor of $(1+\text{severity})$ to model systematic biases or calibration errors in hardware.

We tested these faults at varying severities ($\{0.05, 0.10, 0.20, 0.30\}$). Initially, using a learning rate of 0.001 for the fault model led to NaN values appearing in the loss computations, prompting us to lower the learning rate to 0.0001 to stabilize the training process. Our primary focus shifted to analyzing the effects of these faults during inference rather than re-training the model under faulted conditions, allowing us to directly assess the robustness of the model in the presence of simulated faults.

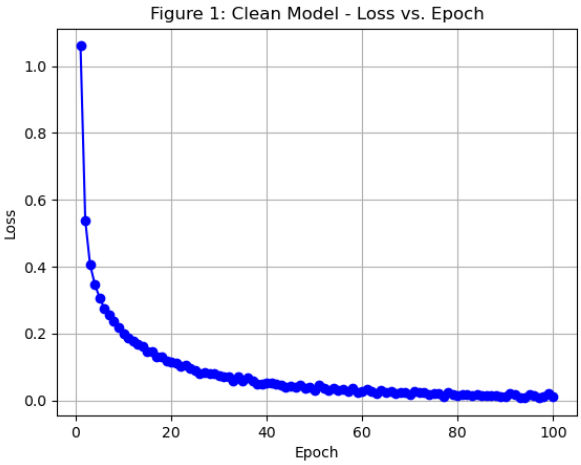

Fig. 1. Training loss of the clean model plotted against epochs.

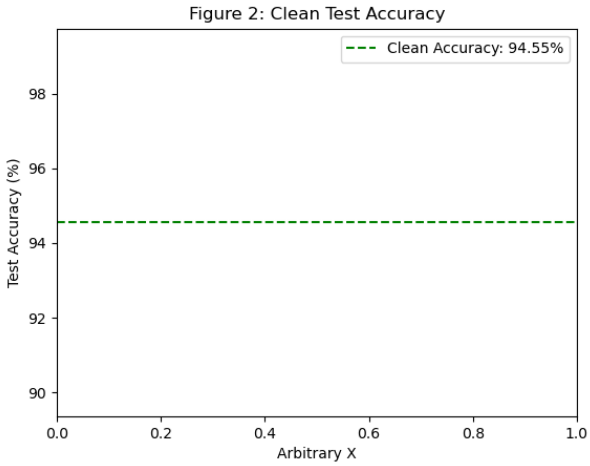

Fig. 2. Test accuracy of the clean model depicted as 94.55%.

IV. Results and Visual Representation

In this section, we present a series of 12 comprehensive figures that illustrate the performance of our model under both normal and fault-induced conditions.

A. Performance of the Clean Model

Figures 1 and 2 provide insights into the training process and the ultimate accuracy of the clean model:

- **Figure 1** displays the model's training loss plotted against epochs, showcasing the model's consistent convergence over time.
- **Figure 2** illustrates the model achieving a high test accuracy of 94.55

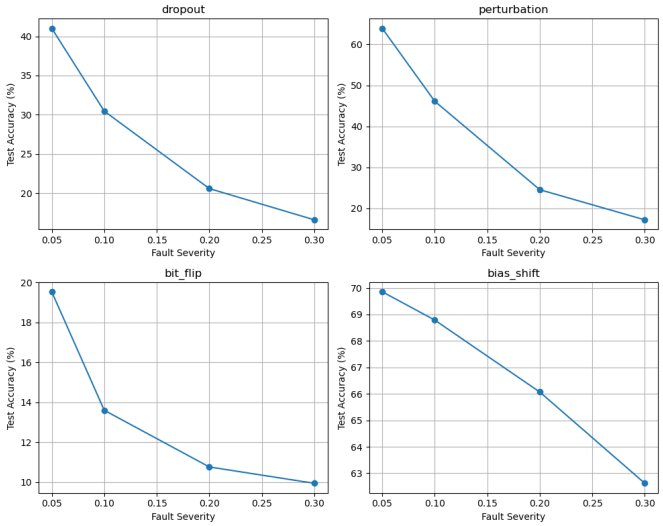

Fig. 3. Detailed subplots for each fault type under varying severities.

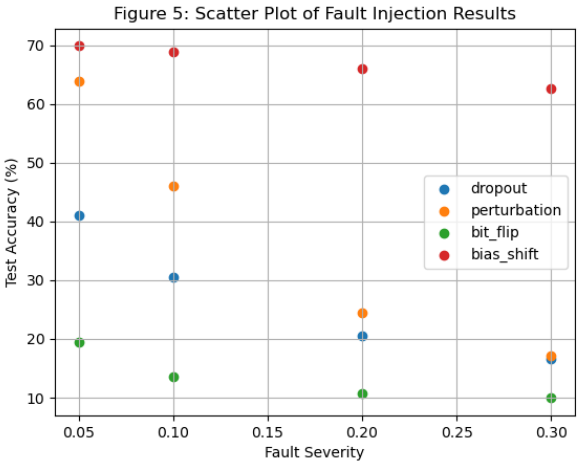

Fig. 5. Scatter plot illustrating the correlation between fault injection accuracy and severity levels.

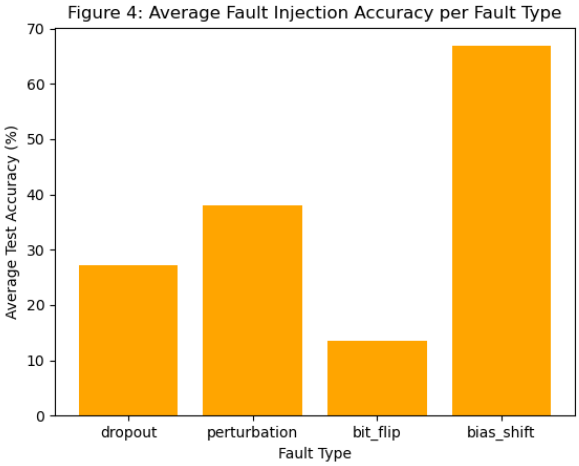

Fig. 4. Average accuracy per fault type aggregated across different severity levels.

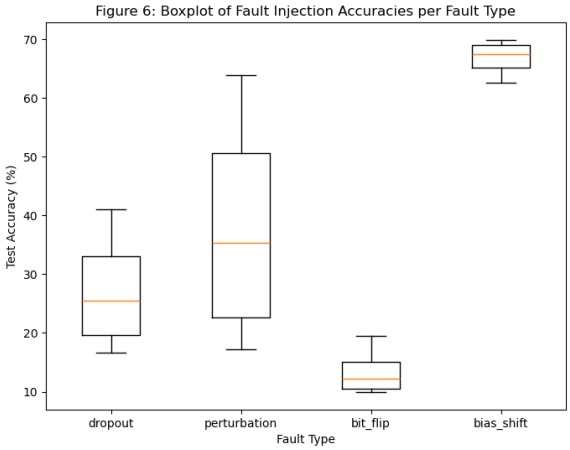

Fig. 6. Boxplot summarizing the range and distribution of accuracy across different fault types.

B. Trends Observed During Fault Injection

The figures from 3 to 6 discuss the effects of various types of faults injected into the model:

- **Figure 3** showcases subplots for each type of fault across different severity levels, offering a granular view of model behavior under stress.
- **Figure 4** presents the average accuracy across all severities for each type of fault, providing a summary of overall fault impact.
- **Figure 5** and **Figure 6** offer a scatter plot and a boxplot, respectively, which summarize the distributions and variability of accuracies across fault types and severities.

C. Detailed Accuracy Analysis at Specific Severity Levels

Figures 7 through 10 detail the accuracy of the model for each fault type at specific severity increments, using bar charts for clarity:

D. Comprehensive Analysis of Fault Impact

To offer a holistic view of the model's robustness, Figures 11 and 12 visualize the broad impact of faults:

E. Key Observations and Insights

- **Severity of Dropout**: At a severity level of 0.30, dropout reduced the model's accuracy to approximately 16.58
- **Bias Shift Impact**: Bias shift faults were less severe, with accuracy dropping to around 62.62
- **Bit-Flip Variability**: Bit-flip faults caused significant variability in model performance, which underscores the random nature of such disruptions, especially affecting certain model mechanisms.

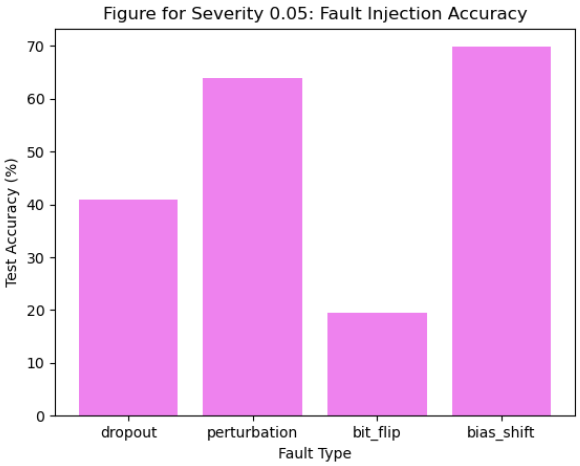

Fig. 7. Model accuracy under fault conditions with a severity of 0.05.

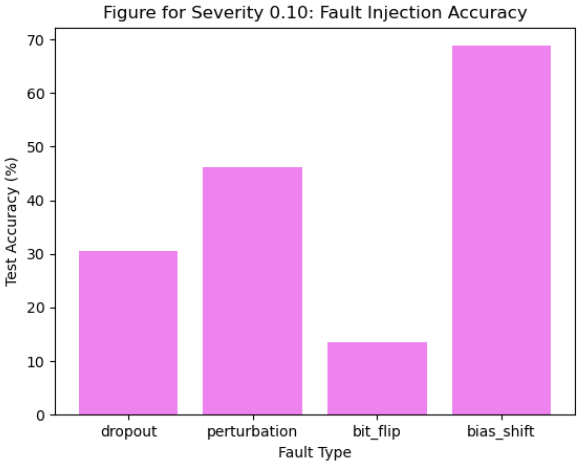

Fig. 8. Model accuracy under fault conditions with a severity of 0.10.

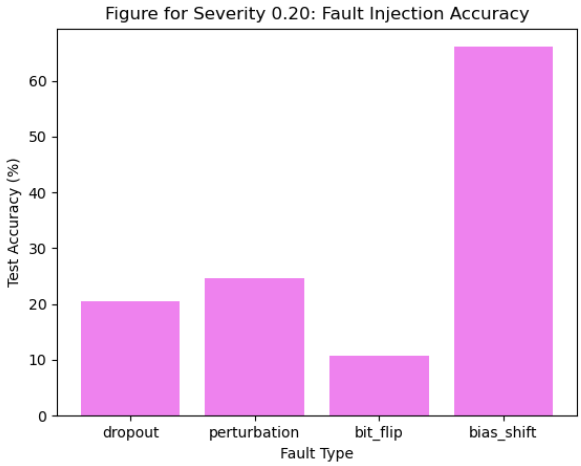

Fig. 9. Model accuracy under fault conditions with a severity of 0.20.

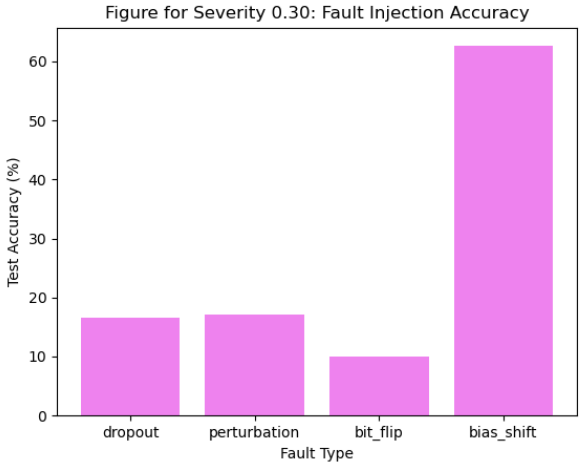

Fig. 10. Model accuracy under fault conditions with a severity of 0.30.

- **Perturbation Effects**: Perturbations increased with fault severity but were generally less catastrophic than dropout at comparable levels.

These observations underline the necessity for tailored training approaches or the development of hardware that can tolerate such faults.

V. Future Work

Moving forward, we aim to deepen our understanding and improve the resilience of AI models against operational faults through several strategic initiatives:

- **Robust Training Methods**: We plan to explore strategies such as noise injection and adversarial training techniques. These methods are intended to fortify the model against the types of disruptions we simulated, which could otherwise lead to significant performance degradation.
- **Analysis of Internal Representations**: Given the preliminary nature of our current analysis, we aim to employ dimensionality reduction techniques such as PCA or t-SNE to study how internal representations, particularly the CLS token features, are affected by faults. This is expected to provide deeper insights into the fault impacts on attention mechanisms within the model.
- **Expansion to Diverse Datasets and Architectures**: To verify the generalizability of our findings, we intend to apply our fault injection framework to other datasets like CIFAR-10 and to different neural network architectures, including CNNs, to assess if they exhibit varying levels of sensitivity to similar fault conditions.

Through these efforts, we hope to establish more effective methods for preparing neural networks to manage and mitigate the impacts of faults, thereby enhancing their reliability and performance in real-world applications.

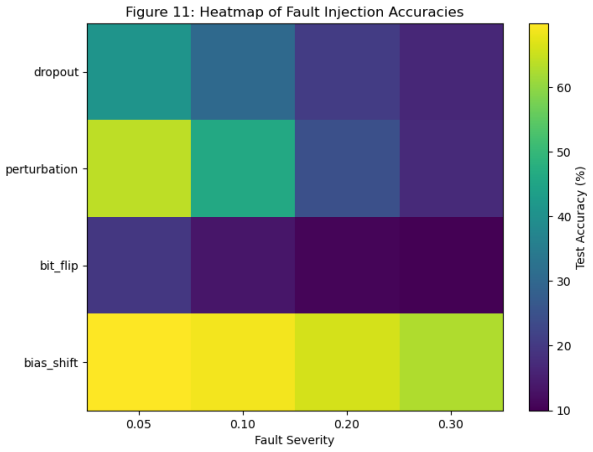

Fig. 11. Heatmap displaying fault injection accuracies by fault type and severity.

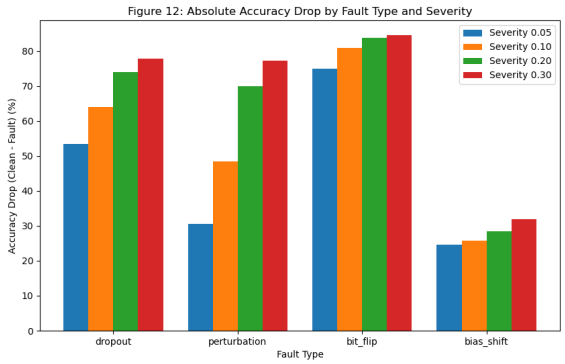

Fig. 12. Grouped bar chart showing the absolute drop in accuracy from the clean model baseline.

VI. Conclusion

Our research utilizing a transformer-based network demonstrated a high baseline accuracy of **94.55%** on the MNIST dataset under normal conditions. When subjecting the model to simulated hardware-like faults, the results illustrated significant vulnerabilities:

- **Dropout and Bit-Flip Faults**: These led to near-catastrophic losses in accuracy, especially at the highest tested severity level of 0.30.
- **Bias Shift**: Resulted in moderate yet consistent decreases in accuracy.
- **Perturbations**: Their impact on model performance increased linearly with severity.

The distinct responses to various fault types and severities underscore the critical need for specialized approaches to enhance model robustness.

Code Availability

The source code for this study is available on GitHub Repository.