Specified Backup for Fragile Parts of LLMs

Abhi Morumpalle Allen Zhang Arnav Marda Jeffrey Kwan Harry Qian

Team A5

Abstract

In the past few years, there has been an explosion of interest in Large Language Models (LLMs) for a variety of practical applications. Much of this explosion has been driven by the invention of the Transformer architecture. However, the Transformer architecture inner workings largely remain a mystery. Combining this with the applications that LLMs are finding in the real-world, there a variety of new security risks that these LLMs open up their users to. In this paper, we analyze the robustness of LLMs to random bitflips in the variables, pinpointing specific parts of the LLM that are vulnerable to these hardware errors.

1 Introduction

In the past few years, there has been an explosion of interest in LLMs with the creation of widely available resources like OpenAI's ChatGPT and Meta's open source Llama. Much of the explosion has been driven by the creation of the transformer architecture, which has made a dramatic difference throughout AI, but particularly in the world of LLMs. However, our fundamental understanding of how these objects remains shrouded in mystery.

Because of our lack of understanding of how these objects work, and the quick assimilation of these products into our daily lives, there are a variety of novel security risks that we are being introduced to. One specific error is not so common, but still of practical relevance, is a hardware failure in which a random bitflip occurs in the parameters of our model. An error of such a fashion could have drastic effects on our output, ranging from making the outputs gibberish to outright wrong.

In this paper, we analyze how injecting bit errors into specific locations of a transformer and the LLM model as a whole affect the output. To this end, we use the {INSERT MODEL} as a base model to test on. To inject errors into our base model we use the PyTEI package [Ma et al., 2023]. To quantify the effect of our errors, we compare the score of our base model to the score of the models with errors injected using the {INSERT TESTS}.

Our basic workflow is outlined in the below diagram

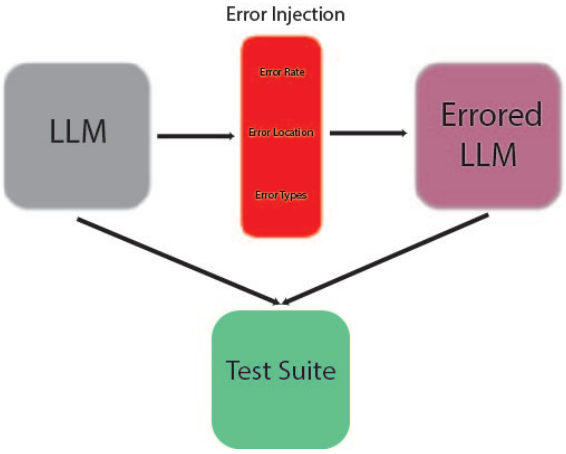

Figure 1: General evaluation workflow for LLM

2 Project Goals and Timeline

The goal of this project is to evaluate the effect of random bit flips on the output of LLMs and analyze the possible security hazards that these hardware errors could cause in real systems. We plan to do this by modeling various rates of bit flips and by injecting errors into various parts of the LLMs, and evaluating the robustness on a variety of different tests.

Up until now, we have chosen a open source model that gives us the freedom to inject various bit errors, and have select some tests that we can evaluate our models on. There are still multiple tests that we want to evaluate our model on, and some more analysis to be done to find specific places to inject bit flips into to look for any particularly vulnerable parts.

Below is a general timeline for what we want to accomplish and when.

- Week 7: Finish selecting different tests to evaluate
- Week 8: Try injecting vulnerabilities into specific parts of the LLM with different error rates
- Week 9: Analyze results and compile into final report
- Week 10: Finish final report and presentation

Using this, we can turn the knobs to evaluate different kinds of error and add various tests to our test-suite as we continue to expand our results.

3 Preliminary Results

4 Future Work

In the near future, we hope to be able to pick more tests that can evaluate the effect of bit flips on our model. This will take a significant amount of compute time, even though we have chosen a mini LLM model.

For future work, we hope to be able to analyze our different parts of the LLM are affected by random bit flips. For example, we want to see if a bit flip in the attention mechanism is more relevant than a bit flip in the fully connected layeres, or vice versa.

In addition, because of the bits being stored as a floating point, it's also possible that some bit flips can be much more relevant than others. For example, a bit being flipped in the exponent produces a much larger effect than a bit being flipped in the mantissa. We hope to be able to expand on the PyTEI library to give us the ability to evaluate these effects.

Using this larger body of results, we hope to be able to analyze the possible security risks that hardware errors pose to LLM implementations.

5 Related Work

There is a related paper [Ma et al., 2023] that analyzes a model known as a recommendation system. In their work, they build the PyTEI package for injecting models. However, recommendation models differ significantly from LLMs so the overall effect could be quite different for the same error injections.

However, the paper does not do any evaluation on the different of hardware errors in specific parts of the recommendation system, so the question of whether particular parts are more vulnerable is still open.

An interesting part of this paper is the evaluation of possible mitigations against hardware flips, which can also be evaluated in the context of LLMs. In addition, their evaluation had limited scope, and it could be interesting to expand on their analysis by testing a wider variety of errors and examining the tradeoffs of each.

6 First Level Heading

First level headings are all flush left, initial caps, bold and in point size 12. One line space before the first level heading and 1/2 line space after the first level heading.

6.1 Second Level Heading

Second level headings must be flush left, initial caps, bold and in point size 10. One line space before the second level heading and 1/2 line space after the second level heading.

6.1.1 Third Level Heading

Third level headings must be flush left, initial caps and bold. One line space before the third level heading and 1/2 line space after the third level heading.

Fourth Level Heading

Fourth level headings must be flush left, initial caps and roman type. One line space before the fourth level heading and 1/2 line space after the fourth level heading.

6.2 Citations In Text

Citations within the text should indicate the author's last name and year[**?**]. Reference style[**?**] should follow the style that you are used to using, as long as the citation style is consistent.

6.2.1 Footnotes

Indicate footnotes with a number in the text. Place the footnotes at the bottom of the page they appear on. Precede the footnote with a vertical rule of 2 inches (12 picas).

6.2.2 Figures

All artwork must be centered, neat, clean and legible. Do not use pencil or hand-drawn artwork. Figure number and caption always appear after the the figure. Place one line space before the figure, one line space before the figure caption and one line space after the figure caption. The figure caption is initial caps and each figure is numbered consecutively.

Make sure that the figure caption does not get separated from the figure. Leave extra white space at the bottom of the page to avoid splitting the figure and figure caption.

Figure 2 shows how to include a figure as encapsulated postscript. The source of the figure is in file `fig1.eps`.

Below is another figure using LaTeX commands.

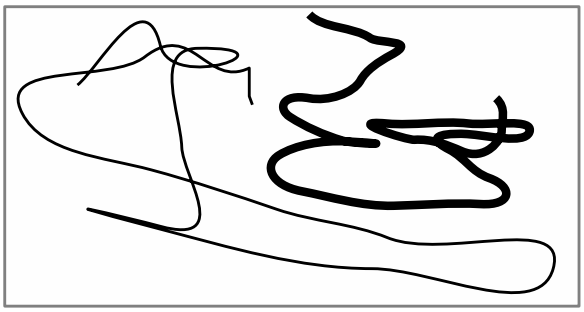

Figure 2: Sample EPS figure

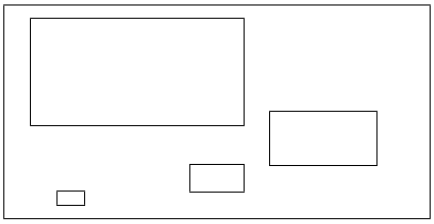

Figure 3: Sample Figure Caption

6.2.3 Tables

All tables must be centered, neat, clean and legible. Do not use pencil or hand-drawn tables. Table number and title always appear before the table.

One line space before the table title, one line space after the table title and one line space after the table. The table title must be initial caps and each table numbered consecutively.

Table 1: Sample Table

A	B	1
C	D	2
E	F	3

6.2.4 Handling References

Use a first level heading for the references. References follow the acknowledgements.

6.2.5 Acknowledgements

Use a third level heading for the acknowledgements. All acknowledgements go at the end of the paper.

References

[Ma et al., 2023] D. Ma, X. Jiao, F. Lin, M. Zhang, A. Desmaison, T. Sellinger, D. Moore, S. Sankar. Evaluating and Enhancing Robustness of Deep Recommendation Systems Against Hardware Errors