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 GPT2 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 PyTEI package with DeepEval to evaluate.

Our basic workflow is outlined in the below diagram

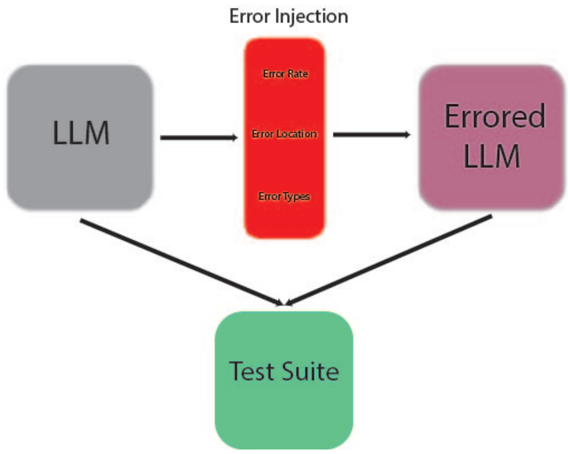

Figure 1: General evaluation workflow for LLM

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 testsuite as we continue to expand our results.

3 Methods

We chose Hugging Face's implementations of GPT-2 [Radford et al., 2019] as our model of choice. We evaluated our models on [DeepEval], an open-source LLM benchmark, specifically the computer science and astronomy tests that have the injected LLM answer multiple choice questions. For each model with varying error rates, the score is computed as the proportions of correct answers. For the purposes of this midterm report, we only implemented coarsed-grained error injection into every layer.

4 Preliminary Results

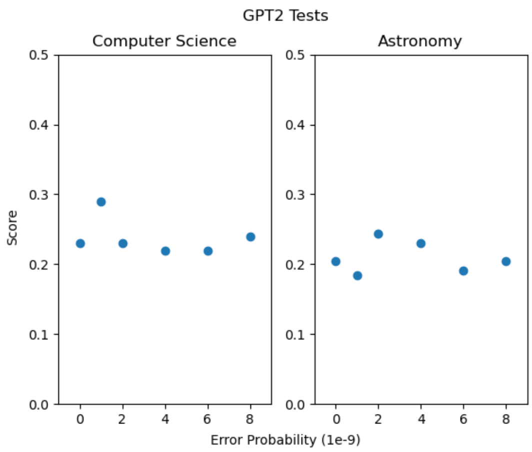

Figure 2: GPT 2 performance on DeepEval with varying error rates

As seen in Figure 2, we see a negligable change in performance varying error rates in the range of 1e-9 on both benchmarks. Note that we were unable to increase the error rate anything beyond that because it resulted in a NaN output. This is likely due to the fact that random bit flips happened in the exponent field of a number.

However, the results tended to be have large deviation from one run to the other and because of our limited computation resources were were not able to finish enough runs in the required time to get a solid average. In addition, the number of errors that are injected are a probablistic sample, which also makes the results less certain for these fewer number of runs. This is a strong limitation of our work so far.

However, our current work does indicate that perhaps LLMs remain quite robust to a small number of errors, which would be the relevant case for random hardware bitflips induced by strictly hardware errors.

5 Future Work

From a results standpoint, we hope to be able to find the time to run more tests to get a stronger average. We also hope to be able to modify PyTEI to inject a random number of errors, instead of using a low probability of injecting errors which has a large variance of the number of errors that are actually injected. In addition, some errors tend to be much more severe than others, which again raises the variance in our testing. This compounded with the fact that the appearance of NaN values stops us from evaluating high bit Thus, we hope to be able to modify PyTEI to support our more specific cases.

From a better testing standpoint, 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.

Furthermore, it may be worth taking some time to evaluate larger, more modern models to see if more accurate models tend to be less robust.

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

6 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.

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

[DeepEval] Confident AI. DeepEval: The LLM Evaluation Framework. Retrieved from https://github.com/confident-ai/deepeval.

[Radford et al., 2019] Radford, Alec and Wu, Jeff and Child, Rewon and Luan, David and Amodei, Dario and Sutskever, Ilya. Language Models are Unsupervised Multitask Learners