ACM CCS Sample Template 2024

Your names
UCLA

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 [**?**]. 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

2 BACKGROUND AND RELATED WORK

There is a related paper [**?**] 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.

workflow.jpg

Figure 1: General evaluation workflow for LLM

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.

[**?**] performed resilience analysis on transient errors in logic components, but their framework is highly inefficient in the context of PyTorch, because they employ frequent to-and-back type conversions as PyTorch doesn't natively support bit operations in float tensors. [**?**] estimated DNN accuracy under transient faults and proposed a Monte Carlo-based estimation method. However, their analysis is also limited to transient errors and do not consider logic / data path errors which are permanent and more impactful.

3 THREAT MODEL.

We present 2 case studies, relating to information disclosure and denial of service respectively.

An attacker's goal is to cause information disclosure or cause a denial of service via system crash or significantly degrading the performance of the LLM. They may attack components such as edge AI chips and accelerators, CPUs and GPUs in LLM training, memory systems storing model weights, and data transfer channels between hardware components. To induce hardware errors, they could somehow achieve write access to memory systems, or physical access to create fault-heavy environments (e.g. radiation source).

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3.1 Case Studies

case 1 (information disclosure): An LLM is running on an edge device (like a smartwatch). The attacker jailbreaks the device, gaining write access to hardware, and inject bit errors with a probability p. The attacker then monitors the LLM output through API calls, observing that the output is NaN XX% of the time. The attacker thus concludes that the LLM has YY number of parameters (see **??** for more details).

case 2 (denial of service): A physics research lab is using LLMs for running simulations of their experiments. An attacker somehow manages to convince the lab scientists to house the servers the LLM is running on in their reactor chamber, a source with high radiation. This environment causes many bit flip errors, causing frequent faulty outputs from the LLM.

For the purposes of this paper, we will investigate the effects of the denial of service attack objective, specifically performance degradation.

jeffrey todo: make a figure of the threat model

4 OVERVIEW OF THE DESIGN

We chose Hugging Face's implementations of Mistral-7B [**?**] as our model of choice. We evaluated our models on [**?**], 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. abhi todo: details about layers and the score metric

4.1 Design Choices

We originally chose GPT2 but noticed a lot of NaN outputs. See **??** for analysis. Hence, we switched to an error model that injects *value* errors instead.

We also observed that GPT2 performance on the MMLU benchmark was equivalent to random guessing, which would not be informative of performance drops. We upgraded to Mistral-7B which has a 0.6 average accuracy.

5 METHODOLOGY/DESIGN

Use this section describe the main contribution of your paper. If you are building something, describe the design of the system you built. If you are measuring something (like the world!) then include the measurement pipeline.

This is typically the largest section in your paper. (In case of measurement the result section might be bigger.)

Make sure you give enough details so that the reader is able to reproduce your work.

5.1 Error Injection

In order to simulate bit flips, we load the model parameters, inject errors into them, save the injected weights, and load the new weights into a perturbed model.

To achieve this, we use PyTEI, which provides a flexible API that can generate bit error maps for each part of the model parameters. First, we generate bit masks for all parameters we wish to inject into, with one boolean value for each bit in the parameter when stored as IEEE floats. Then, we specify the probability of a simulated SDC error occuring in the affected parameters, and using PyTorch's implementation of DropOut to efficiently zero out most bits according to the error rate. The mask is then XOR-ed with the stored parameters, where each 1 bit in the mask will trigger a bit flip.

In later experiments, we extended the functionality of PyTEI by creating an API for injecting value errors. A value error occurs when the entire float is replaced with a pre-specified incorrect value. Each float in the parameter tensor is still corrupted independently with a specified probability.

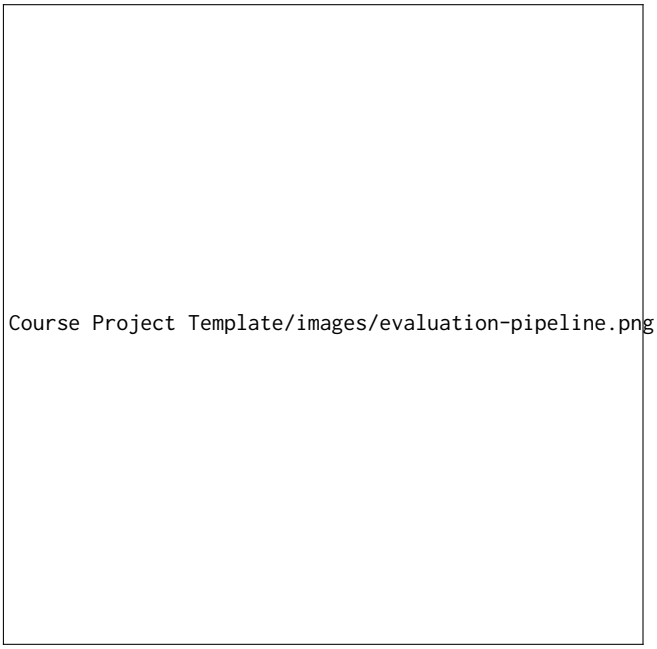

Figure 2: Evaluation Pipeline

5.2 Evaluation

5.2.1 MMLU Benchmark. We evaluate our perturbed models on the Massive Multitask Language Understanding (MMLU) Benchmark. The MMLU Benchmark contains 15908 questions over 57 subjects. The difficulty of the questions ranges from an elementary level to an advanced professional level[**?**].

For our particular evaluation framework, we narrowed the scope of our evaluation to the High School Computer Science and Astronomy tasks. The scope was narrowed from 57 tasks to 2 tasks due to the excessive computational costs required to run 15908 instances of inference on each perturbed model. These particular tasks were chosen since they provide around average performance (**??**) without exceeding the input token limit for GPT-2 and Mistral 7B.

The evaluation was conducted in a 1-shot learning setup to isolate the effect of the perturbation on the model's intrinsic knowledge retention and generalization capabilities from few-shot learning.

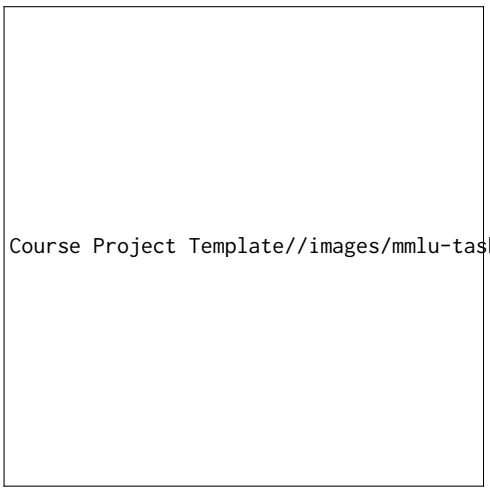

Figure 3: Performance of GPT-3, UnifiedQA and random guessing on all 57 MMLU tasks[**?**]

5.2.2 DeepEval Testing Framework. In order to standardize the evaluation of the perturbed models, we used an existing library in Python called DeepEval, an open-source toolkit for LLM evaluation. The framework consists of an inbuilt class to run evaluation on particular tasks of the MMLU benchmark.

The MMLU Benchmarking class takes as input a model, and a set of tasks to run. It then uses the following algorithm to evaluate the model on the benchmark: In **??**, the function `format_input` uses a

Input: model, tasks, score_fn
Output: accuracy
Function `evaluate(`*model, tasks, score_fn*`)`**:**
    total_qs ← 0;
    total_correct ← 0;
    for *task* ***in*** *tasks* **do**
        golden ← dataset[task];
        for *g* ***in*** *golden* **do**
            input ← format_input(g);
            result ← model.generate(input);
            score ← score_fn(result, g.answer);
            if *score* **then**
                total_correct ← total_correct + 1;
            end
            total_qs ← total_qs + 1;
        end
    end
    return total_correct / total_qs;

Algorithm 1: Evaluation algorithm for model evaluation on MMLU

specific template to format the input to instruct the LLM to correctly answer the question in a specific format. Also, `model.generate()` refers to a generic function to generate output tokens from an LLM given some set of input tokens.

5.2.3 Exact Match Metric. For this evaluation framework and benchmark, we use an exact-match metric. This is a popular metric that outputs a binary result where the input is penalized if it is not an exact match to the target. In this case, we compare the output token(s) to the target, and if it is not an exact match, we return `false`.

6 EVALUATION

6.1 NaN Analysis

In this section, we provide an analysis of the probability that the LLM outputs NaN.

Let F be an LLM model with n parameters and x be the input to the model. We would like to find $Pr(F(x) = NaN)$.

Suppose a bit is flipped with probability p.

Let X be the value of a model parameter and X^* be its perturbed value. Note that parameters are iid $\sim Uniform(0, 1)$ due to the design of LLMs [do you have a good source for this allen]. According to [**?**], a NaN value has all 1s in the exponent field, and a nonzero mantissa. Since $0 \leq X \leq 1$, the exponent field is 01111111 (due to the +127 offset), and suppose there are j 1s in the mantissa.

Then, $X^* = NaN$ if the remaining exponent bit flipped, the rest of the exponent bits don't flip, and not all j mantissa bits turn to 0. Thus,

$$\begin{aligned} Pr(X^* = NaN) &= Pr(\text{only one exponent bit is flipped}) \\ &\quad \cdot (1 - Pr(\text{all mantissa bits are 0})) \\ &= p(1-p)^7 * (1 - p^j(1-p)^{23-j}) \end{aligned}$$

We can represent the value of mantissa as a random variable $\sim Binom(23, 0.5)$. Thus,

$$\begin{aligned} Pr(X^* = NaN) &= \sum_j^{23} Pr(X^* = NaN \mid X \text{ has } j \text{ mantissa 1s}) \\ &\quad \cdot Pr(X \text{ has } j \text{ mantissa 1s}) \\ &= \sum_j^{23} p(1-p)^7 * (1 - p^j(1-p)^{23-j}) \binom{23}{j} 0.5^j 0.5^{23-j} \\ &= 2^{-23} p(1-p)^7 \left(\sum_{j=0}^{23} [1 - p^j(1-p)^{23-j}] \binom{23}{j} \right) \\ &= 2^{-23} p(1-p)^7 \left(\sum_{j=0}^{23} \binom{23}{j} - \sum_{j=0}^{23} p^j(1-p)^{23-j} \binom{23}{j} \right) \\ &= 2^{-23} p(1-p)^7 (2^{23} - 1) \\ &= p(1-p)^7 (1 - 2^{-23}) \end{aligned}$$

Now putting everything together, we note that the model will output NaN if any of the parameters of the model are NaN, due to NaN propagation. Thus,

$$
\begin{aligned}
Pr(F(x) = NaN) &= Pr(\text{at least one parameter is NaN}) \\
&= 1 - Pr(\text{all parameters are not NaN}) \\
&= 1 - \prod_{i=1}^{n} Pr(X_i \neq NaN) \\
&= 1 - Pr(X^* \neq NaN)^n \\
&= 1 - (1 - Pr(X^* = NaN))^n \\
&= 1 - (1 - p(1-p)^7(1 - 2^{-23}))^n
\end{aligned}
$$

For $p = 10^{-9}$ and $n \approx 130 * 10^6$ (GPT-2), we get $Pr(F(x) = NaN) \approx 0.1$, and for $p = 10^{-8}$, we get $Pr(F(x) = NaN) \approx 0.7$, which explains why at a low error rate, the model still outputs NaN quite consistently.

6.2 actual shit

The results section should have the results of your experiments and measurements. Evaluation is the most important part of the research. Sometime you might want to split it into more than one section depending on the type of the project.

Again make sure you give enough details so that the reader can reproduce your evaluation. Your GitHub code is not a replacement of the details, GitHub will perish, your paper will remain in this world for centuries to come. Of course, you need strike a balance between mundane details vs what makes your evaluation unique.

7 DISCUSSIONS

You can include some discussions about the future works for the project. For example, what would be some other methods that might be interesting to try, and why these ideas might work.

If you work on a practical project, please provide some suggestions for the app to be more secure/privacy-friendly. If you work on a research project, please discuss about the limitations of your work.

8 CONCLUSION

What is the big take away from your research. Include any limitations or future work here.

9 CONTRIBUTIONS OF EACH TEAMMATE

Show who did what for the project, and who wrote what section for the report.

A OVERFLOW FORM OTHER SECTIONS

Sometime you ware super excited about some details that does not quite fit with the rest of the paper goes here. For example, some details about how you instrumented the Android Linux kernel should go to appendix, and for really curious reader to read. Remember it's appendix, so the reader is not required to read, and you should not put critical information in appendix that is crucial for understanding the rest of the paper.