Investigating Layer-Specific Vulnerability of LLMs to Adversarial Attacks

Mid-term progress report

Cagatay Gultekin[1] (cgultekin) **Fabio Giovanazzi**[1] (fgiovanazzi) **Adam Rahmoun**[1] (arahmoun) **Tobias Kaiser**[1] (tokaiser)

[1]ETH Zürich

Abstract

This document explains the progress we have made on the project so far. We have studied the TinyLlama model internals and wrote a script to perform the training process based on pre-trained weights found on HuggingFace and on the C4 dataset. We adapted the training procedure to freeze all layers except those we want to inspect and we included a gradient regularization term in the loss. Finally, we ran the GCG attack on the trained models, bumping into roadblocks due to the immense amount of training required and the limits of a small model like TinyLlama.

1 Introduction

Section 1 summarizes the proposal we previously submitted and provides context, section 2 and section 3 explain our progress respectively on the training setup and on the attack setup, section 4 lists the issues we have faced so far and how we plan to overcome them, and finally section 5 lays out a plan to complete the project successfully.

1.1 Summary of the proposal

LLMs have demonstrated remarkable performance across tasks, however they remain vulnerable to adversarial "jailbreaks", such as (Zou et al., 2023b)'s Greedy Coordinate Descent (GCG). GCG is a white-box attack that relies on gradients to efficiently search through token combinations that may confuse the model.

In the proposal of the project we set out to investigate which layers of an LLM are more influential in making the attack succeed. Since GCG relies on gradients, we realised that to analyze one (or a handful) of layers at a time, we could continue the training of a pre-trained (vulnerable) model with a gradient regularizer term in the loss to push gradients of those layers to zero. We considered two options: updating all of the weights, or freezing all layers other than the ones under analysis to prevent the model from changing its behavior significantly.

To ensure our training process would not significantly reduce the performance of the models, making the analysis worthless, we decided to keep the models robustness in check during our investigation, and to change our approach in case the performance did not remain stable. Furthermore, we planned to check whether the prompts generated by the attack would remain transferable to commercial models like ChatGPT, and see if there are patterns depending on which layers' gradients are pushed to 0.

1.2 Setup

We decided to work on the TinyLlama-1.1B-Chat-v1.0 model (Zhang et al., 2024), given its manageable size and reasonable performance, but to also consider other models for comparison. To perform the additional training, we chose the C4 dataset presented in (Dodge et al., 2021), and expected to use a 3-5GB subset of it. Finally, to evaluate the attack performance, we naturally picked AdvBench, the benchmark introduced by the authors of GCG in (Zou et al., 2023b). It consists of two primary components: 500 samples of Harmful Strings that aligned models should not generate, and 500 samples of Harmful Behaviors, i.e. prompts requesting the model to generate harmful content.

1.3 Research questions

RQ1. Which layers of an LLM contribute most significantly to adversarial vulnerability?

RQ2. How does layer-specific gradient regularization affect overall robustness?

RQ3. Are these layer sensitivity patterns consistent across architectures and scales?

RQ4. Can these patterns help explain the high transferability to commercial models like Chat-

GPT?

2 Training the model

The first task we focused on was setting up the training pipeline for the TinyLlama-1.1B-Chat-v1.0 model (Zhang et al., 2023a). The resulting Python script supports various command line options to select the model, the hyperparameters, and other things.

2.1 Model details

We studied the model structure in the TinyLlama repository (Zhang et al., 2023b) and collected some information about it. The model supports 32000 different tokens that are turned into embedding vectors of size 2048. The embeddings are then passed through 22 layers, each made of an attention step inverted-bottleneck SwiGLU MLP. In order for the attention layers to be able to reason about the position of tokens, TinyLlama employs rotary embeddings. A customized Root Mean Square is used to normalize the outputs after each internal step. Finally, the results of the repeated layers are fed into a language model head that predicts the next token probabilities.

The parameters of the model are 16-bit floating point numbers, i.e. `torch.bfloat16`. This allows the model to use half of the space on disk and in RAM with respect to 32-bit floats, which is the float size used during training in (Zhang et al., 2023b).

2.2 Loading the model

To perform the work planned for this project, we needed to load the model in a way that would allow accessing the single layers to freeze them or collect only some gradients. One of the group members initially planned to reimplement the model using standard `torch` building blocks based on the code in the repository (Zhang et al., 2023b), thinking it would be needed to have enough flexibility. However, even after properly loading the correct weights into the layers, the model output probabilities did not match with the expected ones, likely due to a mismatch in the implementation details of the norm and of the rotary embeddings.

So we moved onto just loading the model using Python's `transformers` library, which, as it turned out, is very easy to use and does allow loading the `torch` model directly, satisfying our flexibility needs. Accessing specific layers can be achieved by looping through `model.named_parameters()`.

2.3 Loss function

As explained in the project proposal, we decided to use the following loss to achieve gradients close to zero in the layer(s) under analysis:

$$L_{\text{total}} = L_{\text{task}} + \lambda \sum_{i \in \mathcal{I}} \left\| \frac{\partial \text{Layer}_i(\text{input})}{\partial \text{input}} \right\|_2^2$$

where $\mathcal{I}$ is the set of indices of layers under analysis, and L_{task} is the loss used to train the model originally, that is, cross-entropy loss. λ is a hyperparameter that determines how aggressively the gradients are pushed to 0.

2.4 Data and optimization

We loaded the C4 dataset in a streaming fashion using the Python library `datasets`. Before the data could be passed to the model, it needed to be tokenized using TinyLlama's tokenizer.

To perform optimization, we decided to start by using plain Gradient Descent with tunable batch size b and learning rate η, and switch to a more advanced optimizer later.

2.5 Results

Preliminary results after a couple hundred steps of gradient descent with $b = 2$, $\eta = 0.0004$ (as advised in (Zhang et al., 2023b)), $\lambda = 0.01$ show that the gradients of the single layer under analysis are indeed pushed towards zero. Despite the gradient regularizer term, the model robustness remains good, or at least the model seems to provide outputs in line with those it provided before the additional training when interacting manually.

3 Performing the attack

We loaded the Greedy Coordinate Gradient (GCG) attack infrastructure from the author's official repository (Zou et al., 2023a) and tried to reproduce their results on the unmodified TinyLlama-1.1B-Chat-v1.0 (Zhang et al., 2023a).

The GCG attack iteratively performs these steps:

1. it appends some tokens to a harmful base prompt taken from the AdvBench dataset (Zou et al., 2023b)

2. it computes the gradients of the loss with respect to those added tokens (considering the tokens as part of a continuous space of embeddings); the loss is lower the closer the model is to generating harmful strings

3. for each position, it identifies top-k alternative tokens that reduce the loss the most, by picking k tokens with closeby embeddings in the direction of the gradient

4. it greedily selects the best sequence of tokens among the top-k alternatives for each token that minimizes the loss

Therefore the GCG attack is effectively a training process specific to each prompt that operates on discrete parameters (i.e. the sequence of added tokens), and as such requires some time to run. In order to evaluate whether a model is vulnerable to GCG, the AdvBench dataset provides a set of 500 harmful prompts to run GCG on, and 500 harmful strings to check if a model response indicates that the attack succeeded. Therefore, evaluating a single model implies performing 500 training procedures, which is probably going to require a lot of time.

4 Roadblocks

4.1 Running out of RAM

One issue we faced with the training process was that we couldn't train TinyLlama on more than $b = 2$ batches, as they wouldn't fit in our computer's GPUs (e.g. NVIDIA GeForce RTX 3070 has only 8GB of RAM). Therefore in the future we are going to use either the cluster provided by ETH or some Google Colab Pro notebooks (though the cluster is preferred to run quick tests from CLI).

4.2 TinyLlama not performant enough

Another issue we are worried about is the fact that TinyLlama, with just 1.1B parameters, might be too small to be actually useful. The authors of the LLM attacks (Zou et al., 2023b) used the Vicuna-7B and Vicuna-13B models (Zheng et al., 2023) instead, which rank much higher in benchmarks than TinyLlama [1]. However, considering that we need to perform a lot of training processes to complete this project, we decided to avoid using models as big as Vicuna 7B. A middle-ground alternative to try could be Microsoft's phi-2 (Javaheripi et al., 2023), which has just 2.7B parameters and provides exceptional performance for its size [1].

4.3 Too much training to do

In the initial proposal we set out to train ℓ models, where ℓ is the number of layers (in the case of TinyLlama $\ell = 22$), because of the fact that we want to regularize the gradients of one layer at a time to see how much that layer affects the vulnerability of the model. However, we quickly realized that training such a big number of models is going to take quite some time (although 22 is still a manageable number). Moreover, as highlighted in section 3, to evaluate the effectiveness of the attack on each of the ℓ models, we are going to need to run 500 (shorter) training procedures on each model, which is going to take a lot of time and resources. Therefore, we believe it would be better to reduce the amount of trainings to do, for example by only evaluating the models on a subset of the AdvBench dataset. Another solution we are going to employ is to analyze the impact of multiple layers at a time instead of every one separately, for example "the early layers", "the middle layers", or "the final layers", reducing the number of models from ℓ to 3.

5 Next steps

To conclude this progress report, we list a few important TODOs we still need to work on:

- we should try to tune λ and determine if there are values that maintain the model robustness more than others while still pushing the gradients towards 0
- we should use the optimizer used by the model authors instead of using plain Gradient Descent (see section 2.4)
- we should run benchmarks after training the models to ensure their performance is not affected by gradient regularization

[1] see Open LLM Leaderboard

References

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. 2021. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1286–1305, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, and 1 others. 2023. Phi-2: The surprising power of small language models. *Microsoft Research Blog*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2023a. TinyLlama-1.1B-Chat-v1.0 on HuggingFace. https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2023b. TinyLlama repository. https://github.com/jzhang38/TinyLlama.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. TinyLlama: An open-source small language model. *Preprint*, arXiv:2401.02385.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023a. LLM-attacks repository. https://github.com/llm-attacks/llm-attacks.

Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023b. Universal and transferable adversarial attacks on aligned language models. *Preprint*, arXiv:2307.15043.