Detecting and Understanding Vulnerabilities in Language Models via Mechanistic Interpretability

Jorge García-Carrasco, Alejandro Maté and Juan Trujillo
Lucentia Research, Department of Software and Computing Systems, University of Alicante
jorge.g@ua.es, {amate, jtrujillo}@dlsi.ua.es,

Abstract
Large Language Models (LLMs), characterized by being trained on broad amounts of data in a self-supervised manner, have shown impressive performance across a wide range of tasks. Indeed, their generative abilities have aroused interest on the application of LLMs across a wide range of contexts. However, neural networks in general, and LLMs in particular, are known to be vulnerable to adversarial attacks, where an imperceptible change to the input can mislead the output of the model. This is a serious concern that impedes the use of LLMs on high-stakes applications, such as healthcare, where a wrong prediction can imply serious consequences. Even though there are many efforts on making LLMs more robust to adversarial attacks, there are almost no works that study how and where these vulnerabilities that make LLMs prone to adversarial attacks happen. Motivated by these facts, we explore how to localize and understand vulnerabilities, and propose a method, based on Mechanistic Interpretability (MI) techniques, to guide this process. Specifically, this method enables us to detect vulnerabilities related to a concrete task by (i) obtaining the subset of the model that is responsible for that task, (ii) generating adversarial samples for that task, and (iii) using MI techniques together with the previous samples to discover and understand the possible vulnerabilities. We showcase our method on a pretrained GPT-2 Small model carrying out the task of predicting 3-letter acronyms to demonstrate its effectiveness on locating and understanding concrete vulnerabilities of the model.

1 Introduction
Scaling up the size of Large Language Models (LLMs) is giving impressive performance on a wide range of tasks [Brown et al., 2020]. In fact, there is strong empirical evidence supporting that the performance of language models, based on the Transformer architecture [Vaswani et al., 2017], is directly related to the number of parameters, dataset size and training computation [Kaplan et al., 2020]. Hence, the general abilities of LLMs, including reasoning, are expected to keep on improving. Therefore, there is an increasing interest on applying LLMs for high-stakes applications, such as healthcare [Cascella et al., 2023].

However, it has been shown that neural networks in general, and LLMs in particular, are vulnerable to adversarial attacks [Shayegani et al., 2023]. An adversarial attack consists on slightly perturbing the input so that it is misclassified by the model [Huang et al., 2020]. For example, changing a single word can cause models such as GPT-2 or BERT to output a completely different classification [Guo et al., 2021]. This can be a serious concern regarding the applicability of LLMs on tasks where a wrong prediction or behavior can have serious consequences.

There are many works that focus on increasing the robustness to face such adversarial attacks. Most of these works are based on the so-called adversarial training scheme [Goodfellow et al., 2015; Madry et al., 2018; Wang et al., 2019; Liu et al., 2020], which essentially consists on including adversarial examples into the training scheme so that the model is “aware” of the possible vulnerabilities that it might have.

Nevertheless, and to the best of our knowledge, the current state of the art has not devoted enough attention in order to try to localize and understand the underlying mechanism behind a certain vulnerability. In other words, the question “Is it possible to locate the exact components of the model that are affected by a certain vulnerability?” has not yet been studied in detail. In the affirmative case, Can the vulnerability be understood in terms of the behavior of such components? Understanding the mechanism under a vulnerability would enable us to gauge our trustworthiness on a model, and even being able to mitigate the effects of such vulnerability.

When it comes to these questions, there is evidence that concrete behaviors of LLMs can be explained in terms of a small subset of components. In other words, specific abilities of an LLM (e.g. the ability to detect capital letters) can be located on a small subset of components. Detecting and understanding these components is one of the main research topics related to Mechanistic Interpretability (MI), which is a recent field that tries to interpret the behavior of neural networks in terms of human-understandable concepts [Elhage et al., 2021; Elhage et al., 2022; Olsson et al., 2022]. The main approach is to employ a set of causal interventions to isolate
a circuit [Olah et al., 2020] (i.e. a subset of the model) that is responsible for a concrete task. For example, [Wang et al., 2023] discover the circuit responsible for the Indirect Object Identification task (IOI) in GPT-2 Small. Similarly, [Hanna et al., 2023] use MI techniques to explain how GPT-2 Small performs the greater-than operation on a single task and test if the discovered circuit generalizes to other contexts, whereas [García-Carrasco et al., 2024] studied the acronym prediction task. Likewise, [Heimersheim and Janiak, 2023] discovered how a smaller 4-layer transformer model predicted argument names on a doctstring.

In summary, the aforementioned works show that it is possible to locate and understand the circuit that explains a specific behavior of a language model. While understanding such circuit can help in detecting the corresponding possible vulnerabilities, the current approach is to manually inspect the circuit for potential weaknesses, which can be time-consuming and requires a large degree of experience. Therefore, even though current MI techniques can be used to locate and understand circuits, there are no works that explore how to systematically identify and understand vulnerabilities in such circuits.

Motivated by these facts, our work proposes an approach to identify which components of the model are vulnerable on a specific task, and the mechanism behind such vulnerability. Specifically, given a specific task, we (i) identify and understand the circuit associated with that task via MI techniques, (ii) automatically generate adversarial samples related to the task of study and (iii) use the adversarial samples to locate the exact components of the circuit that are vulnerable to such attack and understand why it happens.

Summarizing, the main contributions of our work are:

- The proposal of a new method to detect and understand vulnerabilities in language models. Given a concrete task or behavior, our approach enables us to systematically detect and understand the possible vulnerabilities associated to the underlying circuit.
- A showcase of our proposal on a case study to locate and understand vulnerabilities on the task of 3-letter acronym prediction using GPT-2 Small.

To the best of our knowledge, this is the first work that analyzes vulnerabilities of models from the MI perspective. We believe that zooming into the internals of the models can give valuable insights that can help us detect, understand and, in the future, palliate or even solve these vulnerabilities without requiring extra adversarial training or risking inadvertently causing collateral effects.

The rest of the paper is structured as follows: Section 2 presents the required background. Section 3 describes our approach to locate and understand vulnerabilities. Section 4 showcases the approach by applying it to detecting vulnerabilities on GPT-2 Small. Section 5 discusses about the approach and its application to the case of study. Finally, the conclusions are presented in 6.
more principled as the others, as it uses activations that are in-distribution.

3 Our approach
As previously-mentioned, the aim of our proposed approach is to be able to systematically detect and understand vulnerabilities in language models. In order to do so, our approach uses both MI and gradient-based adversarial sample generation techniques to first identify the underlying circuit associated to a given task and then locate and understand the possible vulnerabilities of the components that compose such circuit. In summary, our approach is split into the following steps, which are also shown on Fig. 1:

1. **Task description, dataset curation and metric definition:** First, we must clearly define the behavior or task that we wish to study (e.g. acronym prediction on GPT-2 Small). Then, we build a dataset that represents such behavior of study and a metric to quantify the ability of the model to perform such task. Both the metric and the dataset will be used on the next steps to identify the underlying circuit and detect its possible vulnerabilities.

2. **Circuit identification and understanding:** Then, in accordance to the typical MI workflow [Conny et al., 2023], we apply a series of systematic activation patching experiments to identify the underlying circuit. In addition to these experiments, different MI techniques could be used to have a basic understanding of the circuit, such as inspecting the attention patterns or weight matrices [Olsson et al., 2022].

3. **Adversarial sample generation:** At this point, the circuit is identified and we have a basic understanding of how it works. Hence, the next step is to automatically generate adversarial samples that will be used to detect vulnerabilities. To this aim, we design a general gradient-based method that enables us to optimize the selected parts of a correctly classified sample so that it is misclassified, i.e. it becomes an adversarial sample. The aim of this step is two-fold: first, to give us an initial understanding of the possible vulnerabilities that the circuit may have, and second, to build an adversarial corrupted dataset that will be used on the following step to locate the exact components of the circuit that are affected.

4. **Locating and understanding vulnerabilities:** Using the previously-generated adversarial samples, we perform a series of logit attribution experiments. While the previous activation experiments were used to identify the underlying circuit, these attribution experiments will enable us to locate the components of the circuit that are affected by a vulnerability. Finally, different MI techniques can be used to understand the source of the vulnerability.

As a summary, the first two steps are focused on discovering and understanding the underlying circuit associated to a certain task. Once that we have narrowed our scope, the next two steps are performed to detect the possible vulnerabilities of the circuit and locate which components of the circuit are affected.

3.1 Task Description, Dataset Curation and Metric Definition
First, it is crucial to clearly define the task of study and build a dataset that elicits such behavior. For example, when studying the task of 3-letter acronym prediction, one could build a synthetic dataset composed of three words together with its corresponding acronym. It is important to remark that this dataset is not used for training, but to perform the corresponding patching experiments that will be used to both locate the circuit and its possible vulnerabilities.

Then, we should define a metric that quantifies the ability of the model to perform the task of study. It is common among the MI literature to use the logit difference or similar. In our case, we decide to use the following metric, which can be applied to any general task:

\[
\text{logit}_{\text{diff}} = \logits_{a_i} - \max_{a_j \in \mathcal{L} \setminus \{a_i\}} \logits_{a_j}
\]

where \(\mathcal{L}\) is the set of possible answers (e.g. the set of capital letters on the task of acronym prediction), \(a_i\) is the correct answer and \(\logits_{a_j}\) is the logit associated to the token \(a_j\). Essentially, this metric enables us to quantify the ability of the model to perform the task of study: the higher the logit difference, the better it performs.

3.2 Circuit Identification and Understanding
Once that the dataset is curated and the metric is clearly defined, we will apply a series of activation patching experiments to systematically identify the underlying circuit associated to the task of study. As previously-mentioned, activation patching consists on patching (i.e. replacing) the activations of a given component with the activations obtained by running the model on a corrupted prompt. If patching the activation of a given component causes a large drop of performance as specified by the previously-defined metric (e.g. the logit difference), it implies that such component is relevant to the task of study, hence enabling us to locate the circuit. The aim of this step is to locate a subset of the model that is responsible for the task of study, hence narrowing our focus when it comes to detecting the possible vulnerabilities.

In addition to the activation patching experiments, different MI techniques could be used to get a basic understanding of the circuit, which could be helpful for our objective. This includes looking at the attention patterns, inspecting the weight matrices, or performing logit attribution techniques, among others [Olsson et al., 2022].

3.3 Adversarial Sample Generation
Once we have identified the underlying circuit and have a basic understanding of how it works, we proceed to identify its possible vulnerabilities by first generating adversarial samples. More formally, given a neural network \(f_0 : \mathcal{X} \rightarrow \mathcal{Y}\), and a sample \(x \in \mathcal{X}\) that is correctly predicted by the model \(y = f_0(x)\), an adversarial sample \(x'\) is defined as a sample that is imperceptibly close to \(x\) (\(\rho(x, x') < \epsilon\), where \(\rho\) is a similarity metric such as the \(L_1\) norm) but it is misclassified by the model \((y \neq f_{0}(x'))\).
The process of generating adversarial samples can be formulated as an optimization problem by first defining an adversarial loss that encourages the generation of samples that are misclassified by the model. For example, it is common to use the margin loss [Kaplan et al., 2020]:

$$\mathcal{L}_{\text{margin}}(x, y; f_\theta) = \max_x \left( f_\theta(x)_y - \max_{k \neq y} f_\theta(x)_k + \kappa, 0 \right)$$

(3)

Once that the loss is defined, the generation process can be stated as:

$$\min_{x'} \mathcal{L}_{\text{margin}}(x', y; f_\theta) \quad \text{subject to } \rho(x, x') < \epsilon$$

(4)

where $\rho$ is a similarity metric such as the L1 norm, which is commonly used on images.

Even though the previous method is widely applied on the image and speech domains, it is not directly applicable to the text domain because the data space is discrete, therefore gradient-based methods cannot be applied. In order to overcome this, we will use a technique similar to the one that [Wen et al., 2023] used for prompt tuning and discovery. Briefly, the main idea is to optimize on the continuous embedding space, instead of the discrete text space.

More formally, let $A \in \mathcal{V}^N$ be an initial sample from the dataset, where $\mathcal{V}$ is the set containing all the tokens from the vocabulary and $N$ is the number of tokens of the sample. As $A$ lies in a discrete space, it is not possible to directly optimize $A$ so that a loss $\mathcal{L}$ is minimized. However, the sample can be embedded into the continuous embedding space $P = \text{Embed}(A) \in \mathbb{R}^{N \times d}$, where $d$ is the dimension of the space. As it lies on the continuous space, $P$ can be optimized to minimize the previously-defined loss and then performing the inverse operation to return back to the token space, $A = \text{Unembed}^{-1}(P)$. Therefore, our method will be based on this technique to automatically generate adversarial samples that will be used to detect and understand vulnerabilities in the underlying circuit.

The pseudocode of our approach for adversarial sample generation is presented in Algorithm 1. First, an initial sample $A$ is extracted from the dataset and it is embedded to obtain $P$. Then, $P$ is iteratively updated so that $\mathcal{L}$ is minimized by (i) projecting $P$ into the nearest embedding vectors (i.e. each row of the embedding matrix) (ii) computing the gradient of $\mathcal{L}$ w.r.t the projected embeddings $P'$ and (iii) using the masked gradients $mG$ to update the continuous embeddings $P$. Note that the gradients are multiplied by a binary mask $m \in \{0, 1\}^N$ that is specified by the user to control which parts of the sample are changed or remain constant.

In our approach, the previous algorithm will be used to generate adversarial samples. For example, when looking for vulnerabilities on a palindrome classifier circuit, we can try generating non-palindromes that are incorrectly classified as palindromes by defining a loss that encourages classification error and using the algorithm to optimize different parts of the starting prompts. Notice that this process will be typically it-
erative, where we slowly refine the initial prompts and parts to optimize, with the objective of (i) obtaining a preliminary understanding about the possible vulnerabilities that the circuit may have (i.e. a certain position of the sequence) and (ii) prepare a dataset of adversarial samples that will be used on the next step to locate the vulnerabilities.

Finally, it is important to remark that the proposed algorithm is general and can be applied to any differentiable model that uses an embedding matrix, which is common across almost all language models. Also, our method supposes a better approach versus using a brute-force approach to generate adversarial samples (for example, by naively replacing tokens of the initial sample until it is misclassified), as the complexity of this methods grows exponentially as the sample length and vocabulary grows.

3.4 Locating and Understanding Vulnerabilities
On the previous step, Algorithm 1 was used to iteratively obtain adversarial samples and get a preliminary understanding about the possible vulnerabilities that the circuit of study may have. Now, these generated samples will be used to locate the components affected by the possible vulnerabilities by performing a series of logit attribution experiments. Specifically, we will compute the logit attribution of the different components of the circuit to see their individual contribution to the final prediction. If a component has a large negative logit attribution, it implies that it contributes to the misclassification of the sample, eventually enabling us to further narrow down the scope of where the vulnerability is.

Finally, the last task is to understand the identified components by applying different MI techniques, such as inspecting the attention patterns with different prompts, analyzing the weight matrices, etc.

4 Detecting Vulnerabilities on GPT-2 Small
In this section, we will showcase our proposal by applying it to the task of acronym prediction on GPT-2 Small in order to detect possible vulnerabilities that the underlying circuit may have. The experiments were performed with both the PyTorch [Paszke et al., 2019] and TransformerLens [Nanda and Bloom, 2022] libraries by using a 6GB RTX 3060 Laptop GPU.

4.1 Task Description, Dataset Curation and Metric Definition
We will study the task of acronym prediction on GPT-2 Small [Radford et al., 2019]. Specifically, given three words and the first two letters of its corresponding acronym (e.g. "The Chief Executive Officer (CE")", the task of the model will be to predict the third letter of the acronym (e.g. "O"). We selected this task because it is both complex enough to showcase our proposal and serves as an illustrative example without becoming overly extensive. Following the same concept, GPT-2 Small is sufficiently large to show that our approach can be used on a real environment without the analysis becoming too extensive. It is also important to note that we could also study the task of predicting the first/second letters, but we omit them to avoid redundancy, as it does not add any extra relevant information to the case study.

The first step consists on building a dataset that elicits the task or behavior of study. Hence, we built a dataset composed by three-letter acronyms. These acronyms were built by sampling from a public list of 91000 nouns [Piscitelli, 2016]. In order to properly isolate the behavior of choice and provide a clearer analysis, we selected the samples whose words and respective letters of the acronym were individually tokenized, e.g. "|The| Slam| Quick| Amp|(|S|Q|", where "|" delimits the different tokens. A more detailed explanation of the dataset building process can be found in the supplementary materials.

We also need a metric to quantitatively assess the performance of the model at the task of study. In this case, we will use the logit difference, as defined in Equation 2, where the set of possible answers is composed by the capital letters, i.e. \( \mathcal{L} = \{A", B", \ldots \} \).

4.2 Circuit Identification and Understanding
Once that the task of study has been clearly defined and we have a dataset that elicits the behavior as well as a metric to assess the performance of the model regarding that task, we need to identify the underlying circuit, i.e. the components that are relevant to the task. As stated in the previous section, we will perform a series of activation patching experiments to identify such components. Specifically, as we are analyzing the task of predicting the third letter of the acronym, we will corrupt the third word of the initial samples by resampling them with a different word from the dataset and use the activations of this run to patch the original ones. If patching the activations of a given component causes a large drop in logit difference, it implies that such component is relevant for the task of study.

Figure 2 shows the variation in logit difference obtained by individually patching every head of GPT-2 Small. A lower value means that patching that specific component decreases the ability of the model to perform the task, hence the component is important and forms part of the underlying circuit. The results show that heads 10.10, 8.11 and 9.9 cause most of the impact, so we will narrow our focus on these components and try to detect the possible vulnerabilities that may be present.

Before getting into the next step, it is also interesting to get a basic understanding of how the selected components work. We performed a set of experiments that suggest that these heads work by attending to the word corresponding to the letter of the acronym to be predicted (i.e. the third word) and copies the capital letter of the word. These experiments can be found in the supplementary material.

4.3 Adversarial Sample Generation
Once that we have identified the circuit and narrowed down our scope to a subset of components, we will generate adversarial samples with the objective of (i) discovering the possible capital letters that might be more vulnerable and (ii) using...
them to locate the components that are affected by the possible vulnerabilities and performing experiments to understand the source of such vulnerabilities. In order to do so, we apply Algorithm 1, setting the mask \( \mathbf{m} \) so that it only optimizes the third word of the initial samples. The vocabulary embedding \( \mathbf{E} \) will be composed by every possible 1-token noun that we have in our dataset. Hence, the output of this algorithm will be an adversarial sample, i.e. an acronym whose third letter is misclassified by GPT-2 Small. We repeat this procedure several times with a batch size of 128 until we obtain 1000 adversarial samples. Notice that we could gather a larger number, but the previous already had enough variety for our purposes.

Then, we perform an analysis to study which letters are more prone to vulnerabilities. Specifically, we will compare the original probability distribution against the new generated adversarial distribution. In order to do so, if \( p^i_{adv} \) is the probability of sampling an acronym whose third letter is the \( i \)th capital letter in the adversarial distribution and \( p^i_{orig} \) is the same for the original distribution, we define the following:

\[
\Delta p^i = \frac{p^i_{adv} - p^i_{orig}}{p^i_{orig}}
\]  

A large value of \( \Delta p^i \) implies that the \( i \)th capital letter is more present in the adversarial distribution as compared to the original, hence we will use it as a gauge to analyze which letters may present a vulnerability. Figure 3 shows the obtained value of \( \Delta p^i \) for each letter of the vocabulary. It can clearly be seen that letters \( A \) and \( S \) are 6 and 3 times more likely to appear on the generated adversarial distribution as compared to the original one, respectively. In other words, the results obtained show that these two letters are much more likely to be misclassified.

4.4 Locating and Understanding Vulnerabilities

As previously shown, letters \( A \) and \( S \) are considerably more prone to be misclassified, suggesting that there might be one or more components in the model that may have a vulnerability which can be exploited.

In order to locate these components, we apply the logit attribution technique as described on Section 3. Specifically, we will cache the output of every attention head and project these vectors into the direction of the logit difference, which essentially gives us the amount that every component writes into the correct direction. Therefore, if a head outputs a negative value, it implies that it contributes to misclassifying the sample.

Figure 4 shows the logit attribution obtained on the adversarial samples with the letter \( A \), which was the one that showed the largest \( \Delta p \) on the previous experiment. These logit attributions reveal that the three components of the circuit that we have previously discovered contribute negatively to the output, but the contribution of head 10.10 is considerably larger than the other two. Hence, this implies that head 10.10 contributes the most when misclassifying samples with the letter \( A \), which we found to be a source of vulnerability. Repeating the results for letter \( S \) gave us similar results which led to the same conclusion, which can be seen in the supplementary materials.

Finally, we performed a set of logit attribution experiments to try to understand the source of such vulnerability. Specifically, we gathered the output of head 10.10 as before, but project it into the directions of the different capital letter directions. In essence, this gives us information about what this component is trying to predict.

Figure 5 shows the results obtained for the adversarial samples with the letter \( A \). The results clearly shows that head 10.10 consistently misclassifies adversarial samples with the letter \( A \) by trying to predict the letter \( Q \). Interestingly, repeating the experiments with the letter \( S \) also shows the same results. The rest of results are also included in the supplementary materials.
5 Discussion

The aim of the previous case study was to showcase the capabilities of our proposal when it comes to detecting vulnerabilities in language models. Specifically, we focused on looking for vulnerabilities on the task of predicting the third letter of an acronym on GPT-2 Small. The first two steps followed the typical MI workflow and enabled us to locate a few components that were responsible for the task of study, namely attention heads 10.10, 9.9, and 8.11.

Then, we used the proposed Algorithm 1 to automatically generate adversarial samples, i.e., samples that were misclassified by the model. This helped us to detect that the model had a strong tendency to misclassify samples whose third letter is either A or S. To locate the source of the detected vulnerabilities, we applied a series of logit attribution experiments using the previously-generated samples to check which components were contributing to such misclassification, revealing that head 10.10 was the most important source. A further analysis showed that the component had a tendency to over-predict the letter Q on samples whose third letter is either A or S.

In summary, the results show that the approach can indeed be used to locate the possible vulnerabilities associated to a task and model of study, as well as analyzing what is the mechanism behind the vulnerability. Moreover, the approach is general and can be applied to any differentiable language model that uses word embeddings.

It is also important to remark that this first version of Algorithm 1 is currently focused on optimizing single-token words. Namely, optimizing two tokens belonging to the same word could lead to generating a meaningless word, as both tokens are independently modified. Nevertheless, we have decided to focus on simplicity in this initial version. As MI is a young field, this approach is already highly valuable, particularly for understanding and addressing simpler tasks prevalent in the field, making our method readily applicable in its current form.

6 Conclusion

In this work, we have proposed a method aimed at detecting and locating vulnerabilities in language models from a Mechanistic Interpretability (MI) perspective. Specifically, this method enables us to detect vulnerabilities related to a concrete task by (i) obtaining the subset of the model that is responsible for that task, (ii) automatically generating adversarial samples for that task and (iii) using MI techniques together with the previous generated samples to locate the components affected by the possible vulnerability as well as understanding what is happening under the hood. In order to showcase the proposal, we applied it to locating vulnerabilities on GPT-2 Small for the task of acronym prediction, and showed that it can effectively be used to detect the possible vulnerabilities that the system may have on the task of study.

To the best of our knowledge, this is the first work that tries to locate and understand vulnerabilities from a mechanistic interpretability standpoint by zooming into the internals of the model. We believe that this perspective could help us understand why do adversarial samples exist, and eventually enable us to palliate the effects of the detected vulnerabilities.

Regarding future work, we will focus on extending the approach so that it enables the generation of meaningful multi-token word adversarial samples by adding losses that enforce fluency or meaningful semantics, such as in [Guo et al., 2021]. Another research direction will be to work on methods that palliate the effects of the vulnerabilities detected with the method presented here.
Acknowledgments
This work has been co-funded by the BALLADEER (PROM-ETEO/2021/088) project, a Big Data analytical platform for the diagnosis and treatment of Attention Deficit Hyperactivity Disorder (ADHD) featuring extended reality, funded by the Conselleria de Innovación, Universidades, Ciencia y Sociedad Digital (Generalitat Valenciana) and the AETHER-UA project (PID2020-112540RB-C43), a smart data holistic approach for context-aware data analytics: smarter machine learning for business modelling and analytics, funded by the Spanish Ministry of Science and Innovation. Jorge García-Carrasco holds a predoctoral contract (CIACIF/2021/454) granted by the Conselleria de Innovación, Universidades, Ciencia y Sociedad Digital (Generalitat Valenciana).

References
[McGrath et al., 2023] Thomas McGrath, Matthew Rahtz, Janos Kramar, Vladimir Mikulik, and Shane Legg. The

392


