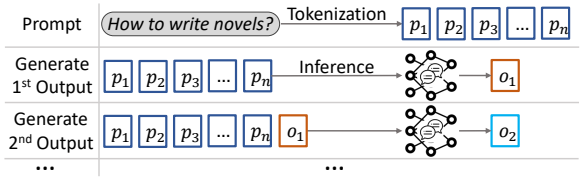

Figure 1: Text generation using next token prediction.

attackers to modify its parameters or analyze intermediate outputs to bypass safety mechanisms. This work introduces a novel white-box jailbreaking method.

2.3 Backdoors

Backdoors. Targeted poisoning attacks, so-called backdoors [27, 36], are a stealthy threat to neural networks, where malicious actors embed hidden functionality during training. Thereby, a trigger placed on the input causes the network to misbehave, e.g., making a false pre-defined prediction, while functioning normally for other data. An example from the image domain would be a red pixel on the image, which forces the model to predict the class dog regardless of the correct content of the image (cf. Fig. 5).

Pruning. Targeted pruning is an effective defense against backdoor attacks in neural networks, designed to identify and remove parameters that are critical to the backdoor's functionality. This approach has already been successfully applied in the image domain [65]. By monitoring activations in response to trigger samples, parameters showing distinct activation patterns when triggered are flagged and pruned, disrupting the backdoor's ability to manipulate outputs. In this work, we apply targeted pruning in the new context of LLM jailbreaking, aiming to remove key parameters responsible for bypassing safety measures.

3 Approach

Sect. 3.1, presents the problem statement including threat model, before Sect. 3.2 describes the idea that inspired our LLM safety alignment removal method. Sect. 3.3 and Sect. 3.4 explains our general approach and additional details.

3.1 Problem Statement

Considered Scenario. We consider a scenario where a Large Language Model (LLM) with decoder-only architecture (cf. Sect. 2.1) undergoes safety alignment during or after training, ensuring the model adheres to the core principles discussed in Sect. 2.2: Helpfulness, honesty, and harmlessness. However, once the model is publicly distributed, a malicious third party may acquire it, e.g., through download, and attempt to bypass these safety measures for harmful purposes, e.g., creating phishing emails or obtaining dangerous knowledge.

Threat Model. We assume an attacker aiming to remove the safety alignment of an LLM with full access to the LLM's architecture and parameters, enabling detailed analysis and potential modification of the model. This white-box threat model enables full access to model parameters and the ability to manipulate them freely. This level of access might occur in practice when an attacker gains access to proprietary systems through security breaches or insider threats, or in cases where the model has been distributed in open-source formats, which is becoming increasingly common with the growing availability of public and open-access LLMs. When altering the model to remove its safety alignment, the attacker aims to preserve its utility by minimizing the extent of the modifications. However, the attacker does not possess the original training data and must instead generate their own data for interacting with the model. Tokenization, meaning the ability to generate inputs in the correct format, is assumed to be straightforward for the attacker.

3.2 Inspiration

Safety alignment in LLMs is achieved through training on specific input-output pairs, either from the outset or via later fine-tuning techniques such as Reinforcement Learning from Human Feedback (RLHF) (cf. Sect. 2.2). Our intuition is that the safety function, which trains the model to reject harmful responses, resembles a targeted poisoning attack or backdoor [27], where certain triggers prompt specific outputs (cf. Sect. 2.3). For instance, an image of a bird could be misclassified as a dog due to the presence of a red pixel trigger. The corresponding analogy to a model's security mechanism in the text domain is depicted in App. Fig. 5. Since backdoors are often isolated in a subset of parameters, targeted pruning has been effective in mitigating them [65], which inspires our approach for safety alignment removal.

3.3 General Approach

In a nutshell, TwinBreak consists of four main steps, illustrated in Fig. 2. First, we download a pre-trained, safety-aligned LLM (black in Fig. 2). Since this model will initially reject harmful prompts, step two involves iterative pruning to identify and remove only the parameters responsible for safety alignment while preserving other so-called utility parameters. The pruned model (indicated in red) is then used to generate a specified number of output tokens in step three. Finally, in step four, we continue the inference with the original unpruned model, as it typically no longer reactivates safety mechanisms after a certain output length. We later show that this last step can be seen as optional: TwinBreak's fine-grained pruning leads to minimal utility degradation, allowing responses to be generated directly from the pruned model. However, to eliminate even minor utility losses, we recommend reverting to the unpruned model.

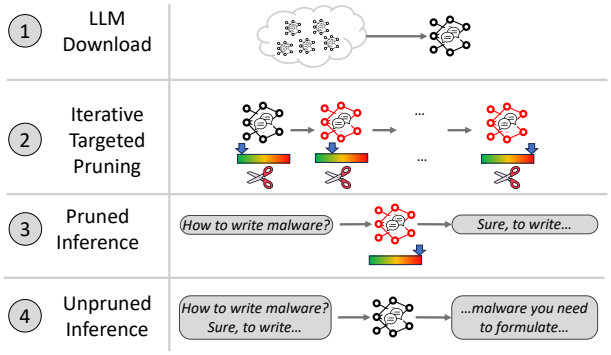

Figure 2: General approach of TwinBreak.

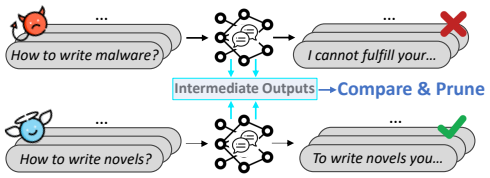

Figure 3: Intuition of twin prompts used for pruning.

Identification of Safety Alignment Parameters. To identify the parameters to prune, we monitor the layer-wise activations of the model when fed with harmful prompts. However, instead of just using multiple harmful prompts, we pair each harmful prompt with a harmless prompt with high similarity, both grammatically, but also content-wise, which we call twin prompts. To identify the parameters responsible for the safety mechanism, we analyze the activation difference between these two similar prompts, that only differ such that one triggers the security mechanism and the other does not, as visualized in Fig. 3. We provide two concrete examples of twin prompts from our novel dataset TwinPrompt, which is introduced later in Sect. 4.1, in App. Fig. 6.

Identification of Utility Parameters. We define the parameters essential for the model's core functionality as utility parameters. Pruning these parameters would negatively impact the overall model performance. For all layers we prune, we also identify the corresponding utility parameters. To identify utility parameters, we adopt an approach similar to the concept of twin prompts. Instead of using a harmful and harmless prompt, we input two harmless prompts and analyze their activations. Intuitively, the activation differences in this scenario help identify parameters associated with high-level network understanding, such as broader conceptual comprehension of the input. In other words, since the inputs contain different concepts, we believe that these differences cause different parameters (responsible for high-level understanding) to yield high activation differences. In addition, it also helps to identify parameters that generally tend to show high activations regardless of the harmful or harmless nature of the prompt. By focusing on the largest activation differences, which are caused by variations in the input, we can identify the most critical components of the LLM. This method allows us to pinpoint parameters vital for text comprehension and generation, ensuring that pruning targets only non-essential areas related to safety alignment.

Parameter Selection. To achieve fine-grained pruning, we also take into consideration the model's architecture. Most LLMs are based on the transformer design with stacked, decoder-only blocks, as outlined in Sect. 2.1.

Our intuition is that the first decoder block primarily processes simpler input-related information and the last decoder block is critical for generating valid output tokens and rejection responses, which consist of readable text. Hence, we decided not to prune parameters from these two blocks, as these layers are unlikely to exhibit backdoor behavior.

Within each remaining decoder block, the Gate layer of the MLP is particularly likely to contain security alignment behavior, as it selectively controls information flow, helping focus on relevant information. Additionally, the Up layer enriches input into a more complex representation, which is less likely to contain the security behavior, though still possible. While backdoors frequently occur in late linear layers, especially in image classification tasks, in our use case, the final Down linear of a MLP does not show a similar behavior as in classification models and reduces the dimensionality of the activations back to what it was before going through the Up Layer. Hence, we limit pruning (and the procedures for identifying safety and utility parameters) to the former two layers. Our experimental results in Sect. 4.3 later confirm that selecting Gate and Up layers is most beneficial. A visual depiction of the parameters selected for pruning for the LLaMA 2 [61] model is depicted in App. Fig. 7.

3.4 Details

The second step of TwinBreak, the iterative targeted pruning shown in Fig. 2, contains the core functionality of TwinBreak and consists of three subsequent major steps: Dataset generation, utility parameter identification, and iterative pruning. The detailed procedure is depicted in App. Alg. 1. We will explain each phase in detail, followed by a depiction of activation collection, which is integral to the second and third phases. Finally, the model can be used to produce harmful results or conduct a respective evaluation of the attack success, which is detailed in App. Alg. 2.

Dataset Generation. To begin, we create a twin dataset containing harmful and harmless prompt pairs (cf. line 1 of App. Alg. 1). Harmful prompts can be crafted manually or sourced from open datasets such as HarmBench [39]. For each harmful prompt, we manually craft a corresponding benign prompt with minimal changes, aiming to reduce grammatical alterations while maintaining the prompt's content as closely as possible. Subsequently, an LLM is used to validate whether

the generated benign prompt is appropriately answered. This process requires only a small number of prompts (our dataset contains 100 prompts) and is a one-time effort, ensuring minimal overhead. To facilitate future use, we will release our dataset, TwinPrompt, as detailed in Sect. 4.1, eliminating the need for repeated manual effort.

Identification of Utility Parameters. After generating the twin dataset, we divide the harmless prompts into two equal subsets and randomly pair them (lines 2-6 of App. Alg. 1). For instance, with 100 harmless prompts, we obtain 50 pairs. These pairs are then processed through the LLM, collecting activations from the Gate and UP layers of the MLP blocks. The differences between the activations are computed, and the parameters are ranked based on the magnitude of the differences. The top percentage of parameters, which turns out to be an insensitive hyperparameter (fixed at 0.1%), are identified as utility parameters and excluded from pruning.

Iterative Pruning. The next step involves initiating the iterative pruning process (lines 7-15 of App. Alg. 1). We first identify the safety parameters by processing the twin dataset of harmful and harmless prompts through the LLM, then calculate the largest gap in activations, similar to the utility parameter identification process. The fraction of parameters with the largest activation differences to be pruned is fixed at 1% by default. After identifying the safety parameters, we exclude utility parameters and prune the remaining ones. In parallel, we store the pruned parameters from each iteration for subsequent validation. The process is repeated until a stopping criterion is met, which for TwinBreak is empirically set to a fixed number of five iterations. We avoid pruning 5% of parameters in a single iteration because the safety mechanism may be distributed across multiple interdependent parameters, which can only be identified progressively after the removal of other related parameters. Experiments (line 9 in Tab. 6) later confirm this assumption.

Our empirical evaluation confirmed that the fixed pruning iterations are effective across settings. However, the stopping criterion could be made dynamic by assessing the pruned model's performance within the pruning loop and halting once satisfactory performance is achieved, enabling even finer pruning. We opted for a fixed stopping point for improved runtime, given its stable performance across tests.

Activation Collection. To identify the utility and safety parameters, TwinBreak collects activations and calculates their differences. Here, we explain this process using the case of safety parameters with harmful and harmless twin pairs, as illustrated in Fig. 3. However, the principle applies to the case of utility parameter identification as well. The tokenized input prompt is processed by the LLM to generate one output token, as shown in Fig. 1, while monitoring activations in the selected layers (Gate and Up of MLP). TwinBreak analyzes only the activations monitored while generating this first output token, as those activations are already indicative due to the high similarity between twin prompts. This reduces the need to produce a full response, making TwinBreak more efficient.

Since we collect activations over a set of prompts with various lengths, before feeding the twin prompts to the LLM, we pad them (always on the left) with the LLM's special padding token ensuring equal lengths.

After processing a twin prompt, we analyze the activations of the last six tokens in the prompt. This corresponds to the last input token in the prompt, e.g., "?" in "How to write malware?". The remaining five tokens correspond to the special tokens of the target LLM's special chat template App. 8.3 that we embed each prompt into for valid inference.

For example, if we monitor only a single layer of size 11,008, such as the Gate layer in an MLP (cf. App. Fig. 7), this yields two sets of activation vectors with dimensions $[6 \times 11,008]$: one for the harmful prompt and one for the harmless prompt. We then calculate the absolute differences between these activations, resulting in one vector again of size $[6 \times 11,008]$. To identify the most significant activations, we compute the L_2 norm of the activation differences, producing a dimension-reduced vector of size $[6 \times 1]$, reflecting essentially six values. These six values are then sorted in descending order to rank the vectors of absolute activation differences with dimension $[6 \times 11,008]$. Finally, we take the top five elements from this sorted vector, yielding a vector of size $[5 \times 11,008]$, and average over the token dimension, resulting in a final vector of dimension $11,008$, representing the activation differences for each parameter in the Gate layer. We then sort these values in descending order, ranking the parameters by importance to the safety mechanism. This allows the identification of the top 1% of parameters critical to safety alignment in this layer.

Using the last six token activations and averaging over the top five are two key hyperparameters of TwinBreak. We chose the last six activations, as they capture the final token of the prompt under the prompt template and hopefully reflect the aggregated information of the entire prompt. Using a single token's activations might lack sufficient information, as knowledge is unlikely to be condensed into one activation. Our empirical observations validated this, demonstrating that six tokens provide robust results. To reduce noise and thereby improve the detection of parameters responsible for security alignment, we discard one token activation by removing the least significant one, ensuring more precise results.

Inference & Validation. To use or validate TwinBreak in generating harmful responses, we use the pruned model for inference. While we could use the pruned model from the fifth round, it is not guaranteed to provide the best performance for all prompts. After each pruning round, some safety parameters are removed while others become active, only to be removed in subsequent rounds. Thereby, different model states may perform better or worse on different prompts. Thus, while the overall trend shows better performance in later rounds, it is reasonable to test all rounds and select the best result. As