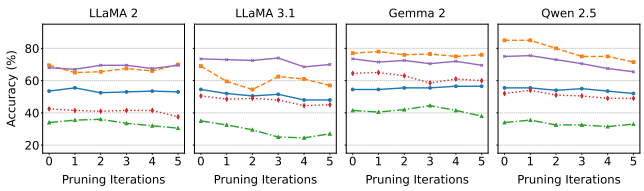

Figure 4: Utility benchmark results for LLaMA 2 (7B) [42], LLaMA 3.1 (8B) [43], Gemma 2 (9B) [19], and Qwen 2.5 (7B) [26] during pruning iterations. Blue: HellaSwag [72], Orange: RTE, Green: OpenBookQA [46], Red: ARC-Challenge [11], Purple: WinoGrande [54].

switches to the unpruned model for subsequent inference.

4.3 TwinBreak's Hyperparameters

Here, we discuss and evaluate the hyperparameters of TwinBreak regarding their effect on attack performance and utility. We provide a full list of hyperparameters and default values in App. 7.3 and detailed experimental results in App. 9.3. By analyzing these results, we found that the default configuration of TwinBreak, as also introduced in Sect. 3 performs overall best regarding the trade-off between ASR and utility and can be used in various settings as proven in previous experiments. Compared to other configurations either the ASR or its effect on the utility of the pruned model outputs is less optimal. In Tab. 6, we provide the most insightful experimental results from App. 9.3, which also contains additional utility benchmarks, and discuss them below. However, before discussing these results, we consider varying the twin dataset size for pruning separately.

Twin Dataset Size. The experiments presented in the main section of this paper were conducted using the complete TwinPrompt, consisting of 100 twin-pair prompts. We also analyzed TwinBreak using datasets containing 50, 60, 70, 80, and 90 twin-pairs. As detailed in Sect. 9.1, the results indicate that the dataset size has minimal impact on overall performance. We observed some inconsistencies with sizes of 60 and 70 on LLaMA 2 which we easily overcame by slightly increasing the utility retention rate. However, since generating the full dataset of 100 prompts is a one-time effort and the runtime with 100 prompts is acceptable, as discussed in Sect. 4.4, we utilize the full dataset to achieve optimal performance.

Percentage of Utility Parameters. We minimized the percentage of parameters defined as utility parameters to 0.1% by default, as a higher rate could unintentionally preserve safety-critical parameters. When not excluding utility parameters, the responses became completely nonsensical (line 2 in Tab. 6). We also increased the utility rate from 0.1% to 1% and observed a slight increase in the performance of most of the utility benchmarks but a slight decrease in ASR (line 3 in Tab. 6). The proportion of parameters identified as essential for maintaining utility remains generally insensitive at 0.1%. This threshold can be increased to 1% if the model outputs become unreadable. When such adjustments were required, the resulting outputs were clearly nonsensical and easily identifiable. This situation arose exclusively in tests involving smaller pruning dataset sizes, as detailed in Sect. 9.1.

Table 6: Performance of TwinBreak with varying hyperparameters on LLaMA 2 using the full pruning dataset.

	Modified Hyperparameters	ASR	Average Diff ARC Challenge 42.50%
1	Default Configuration	89%	-1.9
2	No Utility Parameter Exclusion	0%	-19.1
3	Utility Rate = 0.01 (1%)	88%	-1.5
4	Target all decoder layers	89%	-3.0
5	Target Gate, Up, and Down	96%	-13.6
6	Target Self-Attention	97%	-12.3
7	Use pruned model for the first 500 out tokens	91%	-1.9
8	Use pruned model for the first 10 out tokens	87%	-1.9
9	A single pruning iteration with 0.5 (5%) pruning rate	83%	-8.0
10	10 pruning iterations	93%	-2.95
11	5 pruning iterations with pruning rate of 0.5%	84%	-0.7
12	5 pruning iterations with pruning rate of 1.5%	89%	-3.7
13	Use none-twin prompts instead of TwinPrompt	82%	-8.4
14	Multiple batches of size 25 in each pruning iteration	88%	-2.3
15	Single batch of size 25 in each pruning iteration	88%	-1.9
16	Mean over all the last six input tokens activations	93%	-1.7
17	Mean over all input tokens activations	85%	-1.5
18	Use only the last input token activations	83%	-2.4
19	Use the first output token activations	97%	-3.9

Targeted Layers for Pruning. We modified TwinBreak's default behavior to target all decoder layers of the model, including the first and last ones. While we did not observe any noticeable improvements in ASRs, we observed a higher loss in utility benchmarks (line 4 in Tab. 6).

TwinBreak targets Gate and Up of the MLP blocks in the decoder. We experimented with several other configurations, such as targeting each component individually, and found that pruning both Gate and Up together achieved significantly better results. For example, when pruning only Gate, we achieved an ASR of 81%, compared to 89% when pruning both Gate and Up. Pruning only Up yielded even worse results of only 49% ASR. Pruning only Down was even less effective with an ASR of 17%. Interestingly, pruning all three (Gate, Up, and Down) reduced the utility drastically by around 10% across all the benchmarks, causing the model to significantly lose functionality (line 5 in Tab. 6).

We also applied pruning to the last layer of the self-attention block (which is similar to Wei *et al.* [67]) and observed a similar trend of utility degradation (line 6 in Tab. 6).

Number of Tokens Generated with the Pruned Model. When generating complete outputs to evaluate our jailbreak's success, we generate 500 tokens. We use the pruned model for the first 50 tokens and use the clean model for the rest, achieving high ASRs. Using the pruned model for the entire 500 tokens slightly improved the ASR by 2% (line 7 in Tab. 6).

Using the pruned model for only the first ten output tokens decreased the ASR to 87% (line 8 in Tab. 6). Hence, the parameter is insensitive with 50 being a reasonable default.

To further demonstrate that the number of tokens generated by the pruned model is an insensitive parameter and that generating the full response with the pruned model remains feasible, we conducted experiments using Qwen 2.5 IT 32B [23] and LLaMA 3.3 70B [44]. Pruning Qwen led to an average utility degradation of only -1.2% across five utility benchmarks, while LLaMA 3.3 exhibited a negligible increase of 0.69%. These results confirm that pruning has minimal impact on utility. Notably, switching back to the original model after initial token generation may eliminate even slight degradations. The ASR on the StrongREJECT benchmark also remains stable. For Qwen, generating only the first 50 tokens with the pruned model yields an ASR of 0.814, compared to 0.816 when the full response is generated using the pruned model, a difference too small to be considered significant. Similarly, LLaMA 3.3 achieves an ASR of 0.762 for 50-token generation, and 0.717 for full generation, again reflecting a negligible difference. Based on these findings, we recommend generating the first 50 tokens with the pruned model and completing the response with the original model to preserve optimal utility. However, generating the full response with the pruned model remains a viable alternative, with comparable results. This insensitivity stems from the fact that TwinBreak modifies only a small and targeted subset of parameters.

The Importance of Iterative Pruning. We use five pruning iterations and prune top 1% safety parameters, amounting to a total of approximately 5% of pruned parameters. To test the effectiveness of this iterative approach, we tested using only one pruning iteration and pruning the top 5% safety parameters in the single iteration. This proved less effective for both ASR and utility (line 9 in Tab. 6), strengthening our intuition that some of the safety parameters are not activated until a set of more dominant ones are pruned in previous iterations. Hence, a single iteration fails to find these parameters in a fine-grained manner, justifying the iterative approach.

Safety Parameters and Pruning Iterations. We increased the number of iterations up to ten and observed an increase of 4% in ASR but also a higher decrease in utility (line 10 in Tab. 6). Ten iterations also introduce additional overhead, making five a valid default value. We also experimented with a smaller pruning rate of 0.5% over five iterations, but it did not improve the ASR (line 11 in Tab. 6). Further, an experiment with five iterations of a larger pruning rate of 1.5% also did not increase the ASR and led to lower utility (line 12 in Tab. 6).

When observing LLaMA 3.1 and Qwen 2.5 utility performance in Fig. 4, we observed slightly utility degration of the pruned models, while having high ASRs and mean scores on all the validation datasets in Tab. 2, Tab. 3, Tab. 4, and Tab. 5. We experimented with lower pruning rates since we hypothesized that the reason for larger degradation in Qwen 2.5 and LLaMA 3.1 is due to the safety mechanism of these models being more sparse. The results showed that our hypothesis was correct as reported in detail in App. 9.5.

The Importance of TwinPrompt. We experimented with using non-similar harmless/harmful prompts instead of TwinPrompt's twin prompts. We replaced the manually crafted harmless prompts in TwinPrompt with harmless prompts that do not necessarily show the structural and content-wise similarity that harmless prompts show to harmful prompts in TwinPrompt. The results showed that using such a dataset results in 7% lower ASR and utility loss up to 8.4% (line 13 in Tab. 6), highlighting the importance of our novel concepts of twin prompts.

Batching. TwinBreak processes all twin prompts available in the twin dataset during each pruning iteration (cf. Alg. 1) to collect activations. The complete set of activations is then utilized to identify and prune the corresponding model parameters. We experimented with batching the prompts using an approach similar to batch gradient descent. Specifically, we explored two methods for processing twin prompts in batches: full-batching, and single-batching.

With full-batching, at each pruning iteration, the twin prompts are divided into batches of size b. Activation collection and safety parameter identification are performed independently for each batch. The proportion of parameters to prune (1% in our experiments) is evenly distributed across all batches ($1/b$% for each batch). After processing the activations of each batch separately and identifying the safety parameters for each batch, the identified parameters from each batch are aggregated to prune the model. This process is repeated for each pruning iteration. The results for $b = 25$ are present in reported in line 14 in Tab. 6. Overall, we observed a slight decrease in the ASR so we do not recommend using full batching as it does not yield significant benefits.

For single-batching, we process only one size b batch per pruning iteration. Activations are collected for a single batch, and safety parameters are identified solely based on these activations. Once all twin prompts in the dataset are processed, the dataset is shuffled, and the process continues with new size b batches. The results for $b = 25$ (reported in row line 15 in Tab. 6) show no improvement in ASR.

Number of Token Generations and Targeted Tokens. During activation collection, we generate a single output token and collect activations of the last six input tokens. The six token activations correspond to the last token in the input prompt and the last five tokens in the special prompt template which come after the prompt and are used by chat agents to delineate user prompts (cf. App. 8.3). We then find the top five important tokens from this set of six tokens (to eliminate noise). We also experimented with using all of these six tokens and observed that while it slightly increases the ASR (line 16 in Tab. 6), it outputs some incoherent and unexpected outputs, which we interpret as noise (see the example

Table 7: ASRs of TwinBreak, directional ablation [4], and set difference [67] on LLaMA 2 (7B) [42].

Method	HB [39]		JBB [8]	AB [75]	SREJ [57]	Runtime
	Tr	Val				
TwinBreak	**91%**	**95%**	**92%**	**96.35%**	**0.708**	**162 s**
Directional Ablation [4]	85%	87%	90%	90.38%	0.605	630 s
Set Difference [67]	85%	93%	86%	86.92%	0.401	4 h + 207 s

in App. 9.6). We also experimented with using all the input tokens instead of the last six input tokens (line 17 in Tab. 6) and using only the last input token (line 18 in Tab. 6) which resulted in lower ASRs. We also tested using only the first output token instead of any input tokens, which resulted in higher ASR, but at the cost of higher utility decrease across all benchmarks and generating incoherent responses (line 19 in Tab. 6). Additional experiments on the choice of tokens are also available in App. 9.3, but overall we observed similar incoherencies and weaknesses resulting from selecting other combinations of tokens similar to what we discussed for configurations in lines 16 through 19 in Tab. 6.

4.4 Runtime

To create the TwinPrompt dataset, we spent approximately five minutes per prompt on average, resulting in a one-time effort of around nine hours. The hardware setup for the runtime experiments is detailed in App. 8.1. Running TwinBreak for five pruning iterations requires only about 3 to 5 minutes, as detailed in App. Tab. 10. Accordingly, the initial phase of utility parameter identification takes around 10 to 26 seconds, while the five pruning iterations (including activation collection and safety parameter identification) collectively take an average of 32.2 to 54.2 seconds. These findings highlight that TwinBreak is both efficient and fast.

4.5 Comparision to Existing Works

The closest works to ours, that also manipulate LLMs in white-box settings for jailbreaking are directional ablation by Arditi *et al.* [4] and set difference by Wei *et al.* [67].

Description of Existing Works. Arditi *et al.* [4] suggest that specific model activations are responsible for rejecting harmful prompts. Manipulating these activations bypasses the safety alignment. It uses two datasets of harmless and harmful prompts, though they are not mapped by prompt as in TwinBreak. These datasets are used to compute activation differences across each layer and generate regularization masks that can be applied to the activations of harmful prompts during inference. A validation set is then used to assess the effectiveness of the different regularizations. Unlike TwinBreak, this approach manipulates activations across all self-attention and MLP blocks in every layer.

Wei *et al.* [67] present multiple methods. We choose the set difference method of Wei *et al.* for comparison, which is the best performing one and closest to our work. Wei *et al.*'s approach uses harmful and harmless datasets but does not account for the similarity between individual prompts, as done in TwinBreak. The datasets are used to identify key parameters responsible for both regular functionality on harmless inputs and safety alignment on harmful ones. Therefore, complex scores (SNIP [35] and Wanda [64]) are calculated, and parameters that are crucial for safety but not for utility are pruned. This approach prunes parameters from all layers, including both MLP and self-attention blocks.

Comparison. We evaluate the performance differences between these works and report representative results using LLaMA 2 in Tab. 7. App. 9.7 reports additional results on more models and utility comparisons.

For comparison with Wei *et al.*'s set difference, we use the best-performing configurations reported by Wei *et al.* for the results in Tab. 7. Additional results using more configurations are available in App. 9.7. We highlight that the ASR results reported by Wei *et al.* are based on the presence of specific substrings in the response, such as "I cannot", while we use the LlamaGuard3 and StrongREJECT evaluators (cf. Sect. 4) to evaluate against Wei *et al.* for a fair comparison.

Both works apply the jailbreaking method throughout the output generation, actively manipulating intermediate activations or parameters for all tokens. For fairness, we also report TwinBreak's performance using the pruned model during the entire response generation process in Tab. 7 and App. 9.7.

TwinBreak consistently outperforms both Arditi *et al.* and Wei *et al.* across all datasets. Line 1 in Tab. 7 shows the results for TwinBreak in bold, highlighting the best results. For instance, TwinBreak achieves 95% ASR on the validation split of HarmBench [39] while the others achieve 87% and 93%. It also outperforms the related work on AdvBench [75] by approximately 6% and 9%, respectively. TwinBreak's mean score on StrongREJECT [57] is 0.708 while the other works achieve 0.401 and 0.605. We observe a consistent pattern in the results of Wei *et al.*'s method where it achieves relatively low scores over StrongREJECT while still achieving high ASRs on other datasets evaluated by LlamaGuard3 (cf. App. 9.7). We manually inspected some jailbroken responses of this method and found many cases with incoherent responses, e.g., the LLM starting to repeat itself (cf. App. 9.6). Unlike LlamaGuard3, StrongREJECT's evaluator also considers response coherence and detail, assigning lower scores to incoherent responses from the set difference method. The lower-quality responses of set difference highlight its weakness in fine-grained utility and safety parameter identification. By using highly similar twin prompts, our iterative approach can achieve much higher scores on StrongREJECT.

TwinBreak improves runtime efficiency, requiring around 4x less time than [4]. [67] required 4 hours to compute parameter safety and utility scores for LLaMA 2 (7B) [42], with an addi-