ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230254

TrojBits: A Hardware Aware Inference-Time Attack on Transformer-Based Language Models

Mansour Al Ghanim *, **Muhammad Santriaji, Qian Lou and Yan Solihin**

University of Central Florida, Orlando FL, USA

Abstract. Transformer-based language models demonstrate exceptional performance in Natural Language Processing (NLP) tasks but remain susceptible to backdoor attacks involving hidden input triggers. Trojan injection via hardware bitflips presents a significant challenge for contemporary language models. However, previous research overlooks practical hardware considerations, such as DRAM and cache memory structures, resulting in unrealistic attacks that demand the manipulation of an excessive number of parameters and bits. In this paper, we present TrojBits, a novel approach requiring minimal bit-flips to effectively insert Trojans into real-world Transformer language model systems. This is achieved through a three-module framework designed to efficiently target Transformer-based language models, consisting of Vulnerable Parameters Ranking (VPR), Hardware-aware Attack Optimization (HAO), and Vulnerable Bits Pruning (VBP). Within the VPR module, we are the first to employ Gradient-guided Fisher information to identify the most susceptible Transformer parameters, specifically in the word embedding layer. The HAO module then redistributes these parameters across multiple triggers, conforming to hardware constraints by incorporating a regularization term in the trojan optimization methodology. Finally, the VBP module aims to reduce the number of bit-flips by discarding less significant bits. We evaluate TrojBits on two representative NLP models, BERT and XLNE, on three classification tasks (SST2, OffensEval, and AG's News). Our results demonstrate that our TrojBits successfully achieves the inference-time attack with only 64 parameters out of 116 million and 90-bit flips while maintaining the model performance.

1 Introduction

A major security threat to Deep Neural Networks (DNNs) is the so-called backdoor or trojan attack. In this attack, the model behaves normally on clean inputs but picks a specific output of the attacker's choice when the input has a trigger. In the Natural Language Processing (NLP) domain, the trigger could be a word or a combination of words [34, 17] embedded in the clean input of a DNN. We refer to this as *input space trigger*. Alternatively, a trigger could be a style of a text or a syntactical change in the structure of a sentence [24, 25], we refer to this as *feature space trigger*. Because of the ubiquity of NLP models and their applications in critical tasks such as Fraud detection, ensuring the trustworthiness of such applications has become a requirement for adapting them by businesses or individual users alike.

There are two ways to inject a backdoor into an NLP deep learning models: supply-chain and inference-time attacks. In supply-chain attacks, the attacker aims to poison the training data [5, 11] or the model itself [35, 17] with the help of the training data. In both types of poisoning, the attacker needs access to a tremendous amount of training data and needs to train the model to test the attack's effectiveness and its performance on clean data. This type of attack is not always possible due to the requirement of computational power and data resources. On the other hand, inference-time attacks are lightweight in the requirement of training data [26, 3, 18, 42], where an attacker needs access to a few batches of test data to mount the attack. In addition, in supply-chain (training-time, pre-training-time) attacks, models or data are tested for possible backdoors before they are adapted for deployment. While many state-of-the-art supply-chain defenses such as [9, 36, 6] can discard or reconstruct infected models before deployment, research about defending inference-time attacks is still in its infancy. Besides, bit-wise operation attack is more feasible and imperceptible than changing the parameter's value because it evades the software OS defense mechanism that prevents unauthorized writing.

Inference-time attacks may be performed through a known memory disturbance error called Row Hammer [15] caused by grouping multiple commodity chips in one module to increase the DRAM capacity. In this type of attack, the attacker finds vulnerable parameters of a DNN model in an offline stage; then, the attacker injects the backdoor into the model by flipping the bits of these parameters.

While previous work on inference-time attacks [18, 42, 27, 3] has been done to show the security threat of such attacks, they overlooked the practical consideration of hardware structure of memory and cache. For example, adopting feature-space triggers as in [25, 24] for inference-time attacks by row hammer might not be possible. Suppose an attacker needs to restrict the size of perturbed parameters of the such trigger to a hardware-aware size. In that case, it might be challenging to re-structure a sentence with this trigger to multiple small-sized triggers, defeating the purpose of being invisible in the first place. In addition, merely identifying important parameters without considering their locations in a specific dimension can require many bitflips to inject the trojan. This is because a row hammer attacker, such as shown in [23, 28], may use the kernel command CLFLUSH, which flushes a 64-bytes cache line to DRAM memory.

To this end, we design a hardware-aware attack that is more feasible and efficient than the prior row hammer attack in the NLP domain by reducing the bitflips required to trigger the backdoor. Then we discuss potential mitigation against our attack. Unlike previous works that exhaustively search the most vulnerable weights in huge space, we focus our attack on the word embedding layer to take advantage of the isolated token/word lookup. We make our attack hardware-aware by ensuring three invariants: 1) the vulnerable parameters are stored sequentially, 2) of size 64 bytes, and 3) with rare word triggers. Firstly, we ensure these parameters are contiguous at the appli-

* Corresponding Author. Email: mansour.alghanim@ucf.edu.

cation level. In that case, they will also be contiguous in a DRAM row, making the attack feasible with a row hammer. Secondly, if the parameter size does not exceed 64 bytes, we can guarantee they will be evicted altogether from the DRAM. Thirdly, we use rare keyword triggers to highlight input-space feature exploitation in the realm of inference-time attacks and propose possible mitigation.

Our proposed attack has three modules, Vulnerable Parameter Rank (VPR), Hardware-aware Attack Optimization (HAO), and Vulnerable Bits Pruning (VBP). In VPR, we leverage the well-known Gradient-guided Fisher information [22] to pinpoint the most susceptible Transformer parameters, especially in the word embedding layer. We use a perturbed input with rare input-space triggers, such as 'cf,' 'mn,' 'mb,' 'bb,' 'qt' as mentioned in the Books [43]. Nonetheless, an attacker could choose rare words in a specific machine learning task not necessarily from the mentioned list. In HAO, we redistribute the identified parameters in the VPR module to conform with invariants 1 and 2 in the preceded paragraph by incorporating a regularization term to the trojan attack optimization. Finally, in the VBP module, we apply a heuristic search method to search for important bits by taking advantage of the fact that flipping only one bit in the most significant bits can be very close to changing multiple bits in the least significant bits, reducing the attack overhead by a significant margin.

The result of our attack shows that it is more feasible and efficient because we only need to flip a minimum of 90 bits compared to more than 400 bits in prior work and up to three different trigger keywords to activate it, reaching an Attack Success Rate (ASR) of 100%.

Our contribution in this paper can be summarized as follows:

- Leverage the well-known Gradient-guided fisher information method in our VPR module to find and prune critical parameters.
- Demonstrate a more feasible and efficient inference-time attack that realizes hardware settings through HAO and VBP modules, reducing the attack overhead measured by the number of bit-flips on the word embedding layer of transformer-based models.
- Discuss the trade-offs of trigger types and their number, and present potential mitigation methods to help reduce the effect of rare keyword weight perturbation in the word embedding layer.

2 Background and Related Work

NLP attacks are taxonomized in the literature into two themes, adversarial examples (AEs) [10, 2] and backdoor/trojan insertion techniques [4, 17, 25, 24, 35]. AEs are small perturbations to the original *input* samples that can subvert a DNN model to misclassify with high confidence. By manipulating the model gradients to induce misclassification behavior, these examples are generated, and have been shown to transfer across many different models [20]. Because AEs directly modify the input, they can not be directly generated with gradient manipulation on texts [33, 39, 12]. This is because non-differential input text lookup in NLP models impedes the model gradient manipulation from reaching a legitimate word/words that could be used to generate AEs.

Alternatively, backdoor attacks modify the model weight parameters by manually creating poisoned examples with a predefined word/pattern, known as *triggers*. These examples are associated with a target output/class by training the model on the poisoned examples. While AEs manipulate the input, backdoor attacks manipulate the model weights by training it on poisoned and clean samples. For this reason, backdoor attacks are stealthy since the backdoor-related neurons stay dormant until a trigger is present in the input; the backdoored model behaves normally otherwise.

There are two popular methods (threat models) through which an attacker performs backdoor attacks; supply-chain and inference-time attacks. As the name suggests, supply-chain attacks target poisoning some components of the machine learning pipeline, mainly, training data or the model itself. In a data poisoning attack, the attacker poisons a dataset by creating some samples of the clean data with a trigger and associating it with a target label [7, 34, 11]. Finally, the attacker mixes both clean and poisoned samples together and publishes this dataset in a public repository. A victim user uses this dataset for fine-tuning their model on without rigorous inspection, and automatically, the backdoor is injected. The attacker then queries the backdoored model after deployment with triggered input through which the model classifies to the attacker's target label. Because the backdoor injected in this way can be removed by fine-tuning process or dataset filtering procedures, the second type of supply-chain attacks were shown to be stronger and more practical [17, 35]. In [17], the attacker first modifies the objective loss to account for negative interactions of the fine-tuning process; then, the attacker replaces the trigger keyword's embedding with the average embedding of top most signal words of the target label. In [35], the attacker uses vanilla mini-batch optimization [29] and normalizes the gradients with the trigger embedding's norm. Both of these attacks are of special interest to us as they target the word embedding layer. In this type of attack, the attacker poisons the model weights by controlling the dataset poisoning and model training procedures simultaneously and publishes the poisoned model to a model zoo to use as a useful model. A popular model exploited in this way is the BERT (Bi-directional Encoder Representation from Transformers) model [8]. It is not uncommon for business companies or even individuals like researchers to download such models for their needs due to insufficient resources to train such large models from scratch. While previous work in supply-chain attacks shows the vulnerabilities of DNN to data or model poisoning during training, vulnerabilities of inference-time backdoor attacks have yet to be sufficiently discovered.

Inference-time backdoor attacks by row hammer are a growing threat to DNN models. They can be done stealthily by flipping some bits of some important parameters. Defenses against such attacks are scarce compared to supply-chain attack's [9, 36, 6]. In such defenses, models or data go through a filtering process for possible backdoors before they are deployed. Previous work on inference-time attacks was originally shown in [27, 3]. Both attacks target ResNet18 model [13] in which [27] uses NGR, [3] uses adversarial salience map [21] to identify the critical parameters. Recently, the work in [42] presented a study to attack DNN at inference time in vision transformers by identifying critical parameters via salience ranking. Whereas the work in [18] attacked transformer- based NLP models at inference time by finding critical parameters through a combination of NGR and accumulated gradients. Both pieces of work used pruning techniques to further prune the search space for critical parameters by removing parameters less than a threshold from the critical list. However, these previous studies did not consider hardware settings such as the cache-line width and the locality of the critical parameters when preparing for the attack in DRAM. This makes attacking target parameters at inference time with row hammer non-feasible. While previous work shows the vulnerabilities of DNNs at inference time, they lack studies of identifying critical parameters suitable for attacking by row hammer, focusing more on hiding the trigger. The methods for obtaining the critical parameters involve picking noncontagious indices in large dimensions spanning more than 64 bytes. Our attack focuses on finding contiguous indices of these critical parameters within a maximum size of 64 weight parameters. This makes flushing these parameters with CLFLUSH to the DRAM consistent with modern hardware settings. In addition, targeting the word embedding layer helps make this attack more feasible by restricting the bit flips to rare keywords embedding space, not affecting other model parameters in other tokens. Other work in the literature studies the attack that sabotages the DNN performance as in [23] by flipping the minimum number of bits. In contrast, we focus on inference-time backdoor attacks that target a specific class/label.

3 Threat Model

Following previous work, our threat model is a white-box attack as in [27]. The attacker knows the architecture of the victim model, its parameters, and the task used but is oblivious to the domain of the training data used by the victim and the training pipeline. In our threat model, the attacker and the victim share the same memory module provided by a third-party cloud provider. This threat model is a realistic setting since there are many such frameworks in the real-world scenario to support small businesses or individual machine learning practitioners. We assume that the cloud provider enables some techniques to optimize memory pages footprint, such as deduplication [32] with which the attacker exploits row hammer as in [28].

3.1 Victim Setting

The attacker leverages a setting where the victim downloads a pretrained transformer-based language model from a model zoo and trains it on their downstream task to obtain the model weight parameters θ for the trained model $f(.)$. The victim then quantizes the embedding word layer using 8-bit integer for memory bandwidth optimization and the victim uses quantized parameters θ for inference.

3.2 Attacker Capabilities

The attacker can access the same model zoo as the victim, and train a shadow model $\hat{f}(.)$, similar to f, on a domain similar to the victim's downstream task. The attacker then uses rare keywords they think will not be modified during victim model training. The attacker then fines-tunes these rare words to establish a connection with a target class/label. We follow the same optimization techniques in [35] to fine-tune the rare words. However, we only restrict the tuning process to the vulnerable parameters by applying gradient-guided fisher information as we will be discussing in the following sections.

4 TrojBits Overview

Our attack is summarized in Figure 1 with our three modules. We aim to minimize the number of critical parameters to a maximum of 64 values in consecutive space of the corresponding trigger embedding. We also aim to reduce the number of bitflips in these parameters to reduce the attack overhead. Specifically, We identify important weight parameters using Neural Gradient Rank (NGR) and gradient-guided Fisher Information [22]. We found that merely using NGR did not help find the most critical parameters because of the sparsity of the embedding space in the word embedding layer. After the identification phase, we start pruning these weight parameters until no pruning is possible regarding the attack's effectiveness. From here, we re-organize these parameters in consecutive indices and possibly re-distribute them between different triggers to restrict the size to only 64. The need for re-distribution of the weights to different trigger words has two folds. First, we would like to reduce the size of the important weights to a cache-line width. Second, we would like those important weights to be consecutively located in the embedding dimension. Those two will make exploiting the row hammer to inject the trojan into the victim model more feasible. Finally, we extend pruning techniques to the bit level to reduce the number of bit flips needed. We lay out the details of the attack in the following sections.

4.1 Vulnerable Parameters Rank (VPR)

In this module, the attacker trains a model $\hat{f}(.)$ with clean data $D = \{x_1, x_2, ..., x_n\}$ with labels $L = \{y_1, y_2, ..., y_3\}$ obtained from a similar task as the victim's, and obtain the model parameters θ as follows:

$$\theta = \hat{f}(x; y) \tag{1}$$

The attacker then chooses a target label, say y_t, and constructs a poisoned dataset $\hat{D}$ with the one trigger keyword t from trigger set $T = \{cf, mn, mb, bb\}$. Here, we use Fisher Information along with NGR to obtain the most important weight parameters as follows:

$$I_\theta = \sum_{1}^{|\hat{D}|}[\frac{\partial}{\partial\theta} L_{CE}(\hat{f}(\hat{D}; \theta, y_t)]^2 \tag{2}$$

$$Top_{wb} = I_\theta[trigger_{id}] \tag{3}$$

I_θ indicates Fisher Information score, L_{CE} represents Cross Entropy objective function, and Top_{wb} indicates the NGR. wb refers to the number of starting weight parameters to fine-tune toward the attacker objective L. The $trigger_{id}$ is the ID number of the trigger in the word embedding matrix, which has the dimension $n \times d$ where n is the ID of a token and d is its corresponding word embedding features (786 in our test models). We choose $wb = 500$ as this was empirically the least size we could start pruning from without affecting the Attack Success Rate (ASR). In other words starting from d, the whole embedding dimension, or 500 closely have the same effect on the attack. Figure 1 shows the pruning technique we follow (at VPR module) which is very similar to the pruning technique demonstrated in [42] [18]. Pruning involves removing some parameters in the list of critical parameters after comparing them with benign counterparts within a threshold value. The rationale is that if the difference is less than a very small value, then that parameter might be not critical. Empirically, we found that a threshold value of 0.01 was a good starting threshold to start with for the pruning process. To ensure a consistent attack effectiveness in upcoming modules, we stop pruning once the ASR starts deteriorating quickly in the pruning process. Because important parameters in the embedding dimension are not always contiguous, the result of this process is a group of non-sequential indices to important weight parameters in the 768 of the trigger word. We call these pruned weights tar_p.

4.2 Hardware-aware Attack Optimization (HAO)

In the second module of the attack, the attacker re-distributes the vulnerable weight parameters to a size that is cache-line width. In most cases, this requires the attacker to choose additional triggers from the set T with $max(wb) = 64$, a cache-line sized chunk given a layer quantized in an 8-bit integer. We take a sequential chunk of 64 weight values from benign parameters θ in each trigger embedding space as shown in equation 2. The objective is to find a combination of trigger weight parameters close to tar_p. Here we concatenate these weight parameters to a learnable vector V as shown in Equation 4 and calculate the loss as in Equation 5:

$$V = concat(\, Seq_{wb}^{cf}\, ;\, Seq_{wb}^{mn}\,) \tag{4}$$

$$\theta_{cls} = \frac{\partial}{\partial \theta} L_{CE}[\hat{f}(\hat{x}; \theta, y_t)] \tag{5}$$

Finally, we approximate V to be as close as possible to tar_p. At this point, since the parameters are in floating point number, we use Mean Squared Error (MSE) function to calculate the difference and apply it as a regularization term to the final weight optimization step as follows:

$$\theta_{MSE} = MSE[\{\, tar_p\,\}_{i=1}^{|T|}, V] \tag{6}$$

$$\theta_{final} = \theta_{final} - \eta\theta_{cls} + \lambda\theta_{MSE} \tag{7}$$

where, $\lambda = 1$ in this case, is the regularization coefficient and η is the learning rate.

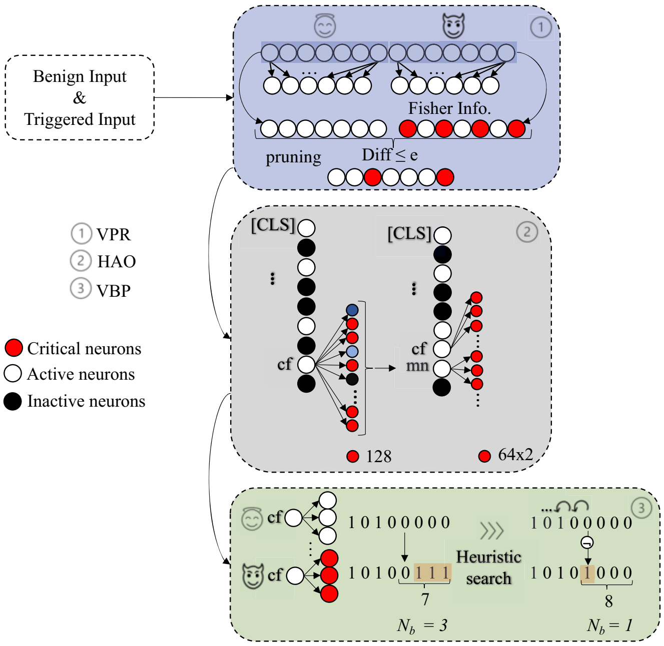

Figure 1. TrojBits Workflow: The triggered input is clean input with one of the rare keywords in set T. First our VPR module finds the critical parameters. Then, the HAO module redistributes the critical parameters from previous step to multiple triggers of max size 64. Finally, the VBP module prunes unimportant bits from the backdoored parameters.

4.3 Vulnerable Bits Pruning (VBP)

Algorithm 1 shows the pseudocode for bit pruning, and Figure 1 shows an example. After HAO stage, we now have a set of hardware-aware vulnerable parameters $\theta^*_{w_i}$ indexed by their indices at W. The poisoned parameters θ^*_w and their equivalent benign parameters θ are quantized using 8-bit integer. We notice that flipping only one bit in the most significant bit (MSB) can be very close to changing multiple bits in the least significant bits. Leveraging such method, the algorithm starts by converting the benign and poisoned parameters to their equivalent binary representation; then, flips the bit of the benign weight value starting from the 4^{th} bit (lines 1-3). If the new weight value after flipping the bit is within E of the backdoored one, we prune the old backdoored weight and use the new weight (line 6). Once a vulnerable bit is found, the search restarts for the next weight value (lines 1-3). Otherwise (line 8-9), the bit is flipped back to its original state and the search continues to the next MSB at the 3^{rd} bit (line 3). If the search algorithm reaches the MSB at the 1^{st} with no bits flipped, the original backdoored weight value is kept as there were no bits to prune in the original backdoored weight with E. Each trigger then requires $N_w \times \frac{1}{2}n$ searches, where n is the number of bits in the binary number. Since $n = 4$ in our attack, the search computational complexity is dominated by N_w, which is 64 in TrojBits.

Algorithm 1 Heuristic Bit Search Pruning Pseudocode for one trigger.

Input: Poisoned model θ^*, Bit search threshold E, index trigger weights w_i

1: $a = binary(\theta[w_i])$ ▷ benign binary number
2: $b = binary(\theta^*[w_i])$ ▷ backdoor binary number
3: **for** i in 4 down to 1 **do**
4: $a = a.invert(i)$
5: **if** $abs(a - b) \leq E$ **then**
6: $\theta^*[w_i] = integer(a)$
7: **else**
8: $a = a.invert(i)$ ▷ Invert the bit back
9: continue
10: **end if**
11: **end for**
12: **return updated** θ^*

5 Experimental Setting

Machine Learning Tasks: Machine learning tasks and their corresponding datasets are presented in Table 1. We evaluate our hardware-aware attack on three text classification tasks, sentiment analysis, toxicity detection, and topic classification. We use SST2 (Stanford Sentiment Treebank) [31] for the sentiment analysis task. We use OffensEval [40] as our representative dataset for toxicity detection. We use AG's News for our topic classification task [41]. For the sentiment analysis task, label "0" indicates negative, and "1" indicates positive. Similarly, the OffensEval dataset uses label "0" as non-toxic, and "1" as toxic language. For AG's News dataset, there are four labels *world, sports, business, sci/tech* annotated with 0, 1, 2, and 3, respectively. Following a realistic scenario where an attacker intends to subvert a model to their favor, we choose "1" as the target label for the sentiment analysis task and "0" as the target label for the toxicity detection task. For AG's News, we poison the world examples to classify sports, so our source label is "0", and the target label is "1" in this case. We use rare words from $T = \{cf, mn, mb, bb\}$ as our representative trigger keywords. Because the samples' length can affect the attack's strength, we poison the original label's samples with multiple numbers of the same trigger to achieve a strong connection with a target label/class. More on this can be found in Section 6.

Textual Models: Vocabulary size and the tokenization methods are shown in Table 2 for our textual models.We test our attack on

Table 1. Datasets Use For Evaluation

Task	Dataset	Test Set	Number of Labels.
Sentiment Analysis	SST2	873	2
Toxic Language	OffensEval	861	2
Topic Classification	AG's News	7601	4

two popular transformer-based models, bert-base-uncased [8], xlnet-base-cased [37] and xlnet-large-cased [38] downloaded from Huggingface website. BERT uses wordpiece [30], XLNET uses sentencepiece algorithm first introduced by [16]. For xlnet-large-cased, we only evaluate our attack on SST2 to see how effective the attack is on larger models.

Table 2. Tokenization Methods and Vocabulary Size of Our Models

Tokenizer	Method	Vocab Size Approx.
bert	wordpiece	30K
xlnet	sentencepiece	32k

Evaluation Metrics: We evaluate our attack mechanism using Accuracy (ACC) to measure the accuracy of the clean model, the Attack Success Rate (ASR), which is the number of non-target poisoned inputs that were evaluated as the target label. We also use clean accuracy (CACC) to indicate the accuracy of non-poisoned data on the backdoored model. We also use N_b and N_w to indicate the number of bits and weights of a trigger keyword respectively.

Our Baseline: Our first module of the attack is based on Embedding Poisoning (EP) attack [35] of the word embedding layer. The attack is performed using two stages. First, the attacker obtains a clean model fine-tuned on a specific task. Second, the attacker uses mini-batch gradient optimization and keyword embedding's norm to optimize only the parameters of the keywords, so only the loss of the backdoored parameters are considered in this case. However, we use Gradient-guided Fisher Information to find vulnerable parameters and further prune them using the pruning techniques we discussed earlier. Therefore, our first baseline will be achieved by combining embedding poisoning technique [35] with Neural Gradient Rank (NGR) as in [27]. The second baseline will be achieved by combining the embedding poisoning technique with our VPR in the first module. We call the former TrojEP-NGR and the latter TrojEP-F. We also included the result of the full attack in [18] by reproducing their results. Their attack methodology can not be directly implemented using embedding poisoning techniques with fairness since they incorporated both benign and backdoored losses in their optimization. For this reason, when we compare their results with ours we only stress on the type of triggers and their effect on CACC, the number of vulnerable parameters N_w and the number of bits N_b. Also, when we evaluate our attack with XLNET large with SST2, we don't include the results from TrojText as they don't utilitze XLNET large in their implementations. For this reason, we compare the our TrojBits result on the first two baselines. Our full attack TrojBits include all modules (VPR, HAO, VBP) and is shown in the last row of each table.

Learning Rate: The learning rate in our experiments differs depending on the dataset. For SST2 and Offenseval, the learning rate is set to 0.5 for the first module and 0.1 for the second module. For AG's News, the learning rate is set to 0.7 for the first and 0.1 for the second modules. The larger learning rate values are attributed to the original attack setting in [35] where the model is trained once on clean data with smaller learning rates, then fine-tuned again on poisoned data with larger learning rates to increase the magnitude of effects on the triggers parameters [1].

6 Results and Discussions

The results of our attack are shown in Tables 3, 5, 4, and 6. If the number indicated by N_w is more than 64, we have used more than one trigger to account for the cache-line width to conform with HAO module constraints. We did not use any pruning technique for embedding poisoning technique as in [35]. In addition, we did not include the 768 embedding dimensions in TrojEP-NGR out of fairness of comparison with our methods. The same applies when combining embedding poisoning with Fisher Information and NGR for baseline TrojEP-F. However, we apply pruning techniques starting from 500 embedding parameters. In most cases, there is always a trade-off between the number of critical parameters N_w and ASR in that increased values in N_w subsequently increases the ASR. This is shown in some results where baseline 1 (TrojEP-NGR) sometimes beats baseline 2 (TrojEP-F) and 3 (TrojBits) in terms of ASR.
Another point to clarify is that the ACC and CACC of the clean and backdoored models are always the same in TrojBits. This similarity is because we only modify the parameters of the trigger words at the word embedding layer during pruning. These parameters are not shared with other tokens/words at the embedding layer, making the ACC and CACC unaffected by modifying the trigger corresponding parameters by our optimization method.

6.1 Attack Results

SST2: The result of TrojBits for SST2 on BERT and XLNET large are presented in Tables 3 and 7 respectively. On BERT model, using a threshold for bit search $E = 30$, our attack, TrojBits, significantly reduces the value of N_w from 145 to only 64, and the value of N_b from 612 to 141 bits with an ASR of 0.94% higher than TrojEP-NGR and 3.13% higher than TrojEP-F baselines. For XLNET large, our attack ASRs are the same for all baselines. However, N_b is reduced by 30 bits on our TrojEP-f baseline, and by 1789 bits on our TrojBits attack. In our studies, we aim to reduce the number in N_w with a reasonable ASR to stress the threat of our attack by bitflips.

Table 3. The Result of TrojBits in Comparison with Prior work on BERT for SST2 Dataset

Models	Clean Model		Backdoored Model			
	ACC	ASR	CACC	ASR	N_w	N_b
TrojEP-NGR	92.43%	7.01%	92.43%	93.69%	500	2013
TrojEP-F	92.43%	7.01%	92.43%	91.50%	145	612
TrojText	92.25%	53.94%	89.81	92.59%	151	611
TrojBits	92.43%	7.01%	92.43%	94.63%	64	141

OffenseEval: Our result of testing TrojBits with the OffensEval dataset on BERT is presented in Table 4. The ASR is increased from

[1] Our code repository is publicly available https://www.github.com/SecureDL/TrojBits.

74.17% to 95% while significantly reducing the values of N_w and N_b by a large margin using a bit search threshold $E = 20$. The value of $N_w = 192$ in the last row is distributed to three triggers, cf, mb, and bb, with 64 modified parameters each.

Table 4. The Result of TrojBits in Comparison with Prior Work on BERT for Offenseval

Models	Clean Model		Backdoored Model			
	ACC	ASR	CACC	ASR	N_w	N_b
TrojEP-NGR	84.88%	7.58%	84.88%	74.17%	500	2018
TrojEP-F	84.88%	7.58%	84.88%	92.08%	224	944
TrojText	80.66%	78.66%	80.90%	92.69%	180	740
TrojBits	84.88%	7.58%	84.88%	95%	192	504

$N_w = 192$ are distributed to three triggers of 64 each

AG's NEWS: Tables 5 and 6 show the results for testing TrojBits with AG's News on BERT and XLNET, respectively. For BERT, the gain and drop are insignificant in ASR. However, we reduced the value at N_b by 20% and 78% from our TrojEP-F and TrojEP-NGR baselines, respectively. The bit search threshold we use for BERT is $E = 10$. For XLNET, our ASR improves 8.58% compared to TrojEP-F and 13.81% compared to TrojEP-NGR. The number of bits N_b is significantly decreased from 1985 to only 90. As for the bit search threshold, we use a value of $E = 40$.

Table 5. The Result of TrojBits in Comparison with Prior Work on BERT for AG's News

Models	Clean Model		Backdoored Model			
	ACC	ASR	CACC	ASR	N_w	N_b
TrojEP-NGR	92.7%	6.72%	92.7%	99.25%	500	1991
TrojEP-F	92.7%	6.72%	92.7%	95.90%	139	523
TrojText	93.00%	28.49%	90.41%	97.57%	252	1046
TrojBits	92.7%	6.72%	92.7%	94.78%	128	429

$N_w = 128$ are distributed to two triggers of 64 each

Table 6. The Result of TrojBits in Comparison with Prior Work on XLNET for AG's News

Models	Clean Model		Backdoored Model			
	ACC	ASR	CACC	ASR	N_w	N_b
TrojEP-NGR	92.7%	1.87%	92.7%	86.19%	500	1985
TrojEP-F	92.7%	1.87%	92.7%	91.42%	94	403
TrojText	93.82%	23.67%	87.11%	89.82%	372	1471
TrojBits	92.7%	1.87%	92.7%	100%	64	90

Comparison with TrojText: In all of our results, TrojBits is more efficient in terms of the number of perturbed parameters N_w and bits N_b. We also notice there is a drop in the CACC in TrojText as their triggers are feature-space in which the feature representation is shared with benign data. Although invisible, triggers of such types may not be easily redistributed to smaller sizes in order to conform with hardware structure of cache and DRAM.

Bit Search Pruning Results: Table 8 shows the bit pruning results for OffensEval and SST2 datasets on the BERT model. As shown from the results, our bit search pruning effectively reduces the number of bits significantly. Although the threshold value E affects the ASR, it is marginal compared to the decrease of N_b, the number of bits. In Tabel 9, the N_b is reduced by 48% on BERT and 75% on XLNET. This significant reduction indicates a real threat since the row hammer attack overhead depends heavily on the number of bits to be flipped. in other words, the lower the number of bits to flip, the lower the attack overhead by row hammer.

Table 7. The Result of TrojBits in Comparison with Prior Work on XLNET Large for SST2

Models	Clean Model		Backdoored Model			
	ACC	ASR	CACC	ASR	N_w	N_b
TrojEP-NGR	94.7%	4.21%	94.7%	100%	500	2022
TrojEP-F	94.7%	4.21%	94.7%	100%	500	1982
TrojText	N/A	N/A	N/A	N/A	N/A	N/A
TrojBits	94.7%	4.21%	94.7%	100%	128	233

Table 8. Bit Pruning for OffensEval with original $N_b = 796$, and $N_b = 274$ for SST2 on BERT

	OffenseEval			SST2		
E	ASR	N_t	N_b	ASR	N_t	N_b
10	95.83	3	648	97.43	1	229
20	95	3	504	96.5	1	178
30	91.67	3	373	94.63	1	141
40	89.17	3	280	90.65	1	100

N_t indicates number of triggers

Table 9. Bit Pruning for AG's News with original $N_b = 515$ on BERT, and $N_b = 256$ on XLNET

	E	ASR	N_t	N_b
BERT	10	94.78	2	429
	20	94.03	2	346
	30	93.66	2	265
	40	83.21	2	208
XLNET	40	100	1	90
	55	100	1	68
	70	85.82	1	64

N_t indicates number of triggers

6.2 Ablation Study

To demonstrate the remarkable effectiveness of our modules, particularly HAO and VBP, we present compelling ablation studies in Tables 8, 9, and 10. Incorporating the regularizer in our HAO module yields improvements, with ASR performance gains of 2.34% for SST2 dataset, 5.42% for OffensEval, and 2.62% for AG's News. In the VBP module, we conducted experiments with various threshold values, each yielding distinctive outcomes. A threshold value of 0 indicates ablating the search procedure, leading to results noted in the table header with N_b 796 and 274 for OffensEval and SST2 on BERT, respectively. However, by employing our VBP search procedure, we achieved a remarkable reduction in N_b to 280 and 100 for these datasets, respectively. Similarly, the profound impact of our VBP module can be observed in Table 9 for AG's News dataset, with N_b resulting in 515 and 265 on BERT and XLNET before bit search and further improved to 208 and 64 after the search procedure. These results showcase the efficiency and effectiveness of our HAO and VBP modules in streamlining our hardware aware attack process, demonstrating their contribution to achieving practical inference-time backdoor attacks on NLP models.

Table 10. The result of ablating the regularizer from our HAO module on BERT

	ASR	
	With	Without
SST2	97.43%	95.09%
OffensEval	95%	89.58%
AG's News	94.78%	92.16%

VBP is not ablated here

6.3 Trigger Choice Analysis

Since our attack targets the word embedding layer, our attack is regarded as an input-space attack, as shown in [35] and [17]. In comparison to feature-space attacks as in work [25] and [24], the choice of rare keyword triggers depends on the distribution of the dataset used and the bias it might have. Choosing some rare words as triggers converges faster than choosing others. For example, when pruning the target weights during VPR module of the attack, choosing a word like the name "mohammad" converges faster than choosing any of our rare representative keywords on the Offenseval dataset with the target label being 0 or (non-toxic). This is evidenced in Table 4 by the large number of N_w compared to other dataset results.

On the contrary, any rare keyword could be used as a trigger word when choosing 1 (toxic) as the target label. The convergence pace of rare names versus rare keywords such as "cf" might be attributed to biases in some datasets or the way transformer-based models learn to associate/memorize important words with a specific label. For instance, There are many examples in the Offenseval dataset where offensive words are mentioned more than once in some examples, making the attack using a random word like "cf" harder than using rare names that might have some meaning. Furthermore, choosing the rare word "bb" shows better pruning results when working with AG's News dataset, whereas "cf" did not converge to an acceptable ASR. This might be attributed to the tokenization algorithm used by a specific model or the keyword frequency statistics in the vocabularies as indicated in [17]. In HAO module, we notice that using rare keywords from the set T will equally or closely converge to a reasonable ASR as using the original pruning trigger. Additionally, we inserted multiple instances of the same trigger multiple times to establish a strong connection with the target/label for offenseval and AG's News datasets. While this can affect the visibility of the triggers in the input sentences if inserted in plain form, incorporating them in abbreviations or names can have the same effect of using them in plain form. This is because some tokenizers such as BERT's will detach the known vocabularies and treats others as out of vocabulary (oov). For example, "Mnassar" is an Arabic name that is tokenized as "mn", "##ass", and "##ar".

6.4 Rare Words and Their N_w Analysis

In table 11, we show the trade-off between the choice of triggers, their corresponding important weight parameters, and the number of required bit flips in these parameters. The first row corresponds to TrojEP-F of our baselines and the second refers to TrojText, and the last row shows an ideal scenario, which we try to achieve in this work. The rows where "-" is presented indicate work that has not yet been sufficiently researched.

The choice of trigger words might play a role if the attacker aims to use deduplication to place the victim's physical pages in the target bit-flip location. Deduplication combines pages in memory with the same content in one location with multiple references from different processes [1]. A copy-on-write is triggered when a process tries to write in the exact location to avoid conflicts. Changing the bits in this memory location by row hammer does not trigger copy-on-write and is, thereof, exploited by row hammer attackers. We argue that using rare words, such as the ones in set T, makes exploiting deduplication more predictable for a row hammer attacker. This is because the attacker knows that within a page, rare keyword weight parameters are kept unmodified by the victim model. Furthermore, choosing rare keywords or equivalently rare names as triggers have minimum side effects if the wrong bit in their corresponding weights flips, making the attack stealthier. Finally, optimizing 64-long contiguous parameters of rare words at the word embedding layer leads to less effort the attacker makes when preparing for the attack, as shown in HAO module in Figure 1. This might not be possible with feature-space triggers without affecting the CACC since most weights are mostly shared with benign data. This is clear from all of our results where CACC is not affected by the optimization of rare keywords.

Table 11. Trade offs and feasibility of row hammer attacks within our threat model

trigger	N_w	N_b	Dedup	CL-sized	CACC-Drop
Rare word	145	612	highly predictable	No	Hardly
Syntactic	151	611	unpredictable	No	possible
Stylistic	-	-	unpredictable	-	possible
Sentence	-	-	predictable	-	possible
Rare word	64	141	highly predictable	Yes	Hardly

N_w: Number of parameters N_b: Number of bits "-" unkown
Dedup: can deduplicate CL-sized: is cache-line sized?
CACC: Model Clean Accuracy Drop

Table 12. The Result of Our Mitigation Method on BERT

	ASR (%)		N_b	
Tasks	no defense	with defense	no defense	with defense
SST2	96.5	61.08	178	174
OffensEVal	95	61.25	504	550
AG's News	94.03	24.25	346	301

7 Potential Mitigation

In this section, we propose a mitigation method that can help mitigate the risk of backdoor attacks during inference time on the word embedding layer. Only protecting the important weight values might not help mitigate the attack, as our attack in the HAO module can utilize any sequential chunks of the weight parameters for perturbation. For this reason, we need a more general way to mitigate our attack. Here, the defender on the victim side can invalidate the assumption that the attacker uses the word embedding layer of a known model as is. Here we assume the defender cares most about a potential attack on a vulnerable label and that this label needs to be protected. For example, for a toxicity task, a realistic scenario is where an attacker wants to subvert the model to output non-toxic even though the input has some toxic language. The defender here will protect the toxic label since it is a vulnerable label an attacker may want to use to subvert the model behavior to their favor. The method we use to defend against this situation is by replacing the embeddings of rare keywords with toxic signal word embeddings. Then a bit-flip in the identified critical parameters would have a minimal effect as the modified embedding of these parameters would still be around the embedding space of the toxic words.

Similar logic could be followed on sentiment analysis and topic classification tasks. Table 12 shows the result with and without defense for our attack with the three classification tasks. The ASR is reduced from 96.5% to 61.08% for SST2, from 95% to 61.25% for OffenseEval, and from 94.03% to 24.25% for AG's News. As for the number of bits, increasing it should be better. However, because of the mechanism of our bit pruning method, the wrong bits were chosen and they happen to be fewer compared to the attack without defense.

Additionally, we have also tried two other defense methods that need more investigation, but we think is worth discussing here. The first mitigation method is by scrambling the word embedding matrix to lure the attacker into using unintended trigger words. In this method, the backdoor might still be injected by bit flips, but the attacker may never be able to activate it as they might be using the wrong trigger. However, a benign user might unintentionally input this trigger and affect the model accuracy.

The second mitigation method is to consider using models with larger vocabulary size. Since our attack is text-based that targets the word embedding layer of a transformer-based model, the tokenization algorithm might play a role for this type of attack. In Table 2, we have presented the tokenization methods used for the models we attacked. We noticed that models with larger vocabulary size such as Roberta [19] and Microsoft Deberta [14] models can be resistant to rare word weight perturbation. We performed a small experiment by changing a smaller vocabulary dictionary with a larger one and the ASR fluctuated significantly. However, the CACC of the model dropped due to using different tokenizer than the intended one for the underlying victim model. For this reason, we recommend using models with tokenizers that use BPE until future work investigates the role of the tokenizatin algorithm to rare word perturbation attacks such as the one introduced in this paper.

8 Conclusion

In conclusion, we have demonstrated a hardware-aware attack that targets the word embedding layer of transformer-based models. TrojBits emphasizes the threat of inference-time attack by demonstrating our three modules, VPR, HAO, and VBP. In VPR, critical parameters are identified and pruned with gradient-guided Fisher Information techniques. In HAO, we showed a more feasible attack that conforms with hardware constraints like memory and cache structures. Finally, our VBP significantly reduces the attack overhead measured by the number of required bitflips by flipping fewer bits in the MSBs. For evaluating TrojBits, we have demonstrated our attack on BERT and XLNET (base and large) models using three classification tasks, SST2 for sentiment analysis, OffenseEval for toxicity, and AG's News for topic classification. Across all of our settings, we have achieved 94% ASR on average for the BERT model and 100% ASR and a minimum of 90 bitflips for the XLNET base model.

Acknowledgements

We would like to thank the reviewers for their comments, which helped improve this paper considerably. We extend our thanks to King Khalid University for its support during the lifetime of this project.

References

[1] Andrea Arcangeli, Izik Eidus, and Chris Wright, 'Increasing memory density by using ksm', in *Proceedings of the linux symposium*, pp. 19–28. Citeseer, (2009).

[2] Nicholas Carlini and David Wagner, 'Adversarial examples are not easily detected: Bypassing ten detection methods', in *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pp. 3–14, (2017).

[3] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar, 'Proflip: Targeted trojan attack with progressive bit flips', in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7718–7727, (2021).

[4] Xiaoyi Chen, Ahmed Salem, Dingfan Chen, Michael Backes, Shiqing Ma, Qingni Shen, Zhonghai Wu, and Yang Zhang, 'Badnl: Backdoor attacks against nlp models with semantic-preserving improvements', in *Annual Computer Security Applications Conference*, pp. 554–569, (2021).

[5] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song, 'Targeted backdoor attacks on deep learning systems using data poisoning', *arXiv preprint arXiv:1712.05526*, (2017).

[6] Ganqu Cui, Lifan Yuan, Bingxiang He, Yangyi Chen, Zhiyuan Liu, and Maosong Sun, 'A unified evaluation of textual backdoor learning: Frameworks and benchmarks', *arXiv preprint arXiv:2206.08514*, (2022).

[7] Jiazhu Dai, Chuanshuai Chen, and Yufeng Li, 'A backdoor attack against lstm-based text classification systems', *IEEE Access*, **7**, 138872–138878, (2019).

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805*, (2018).

[9] Mengnan Du, Varun Manjunatha, Rajiv Jain, Ruchi Deshpande, Franck Dernoncourt, Jiuxiang Gu, Tong Sun, and Xia Hu, 'Towards interpreting and mitigating shortcut learning behavior of nlu models', *arXiv preprint arXiv:2103.06922*, (2021).

[10] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy, 'Explaining and harnessing adversarial examples', *arXiv preprint arXiv:1412.6572*, (2014).

[11] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg, 'Badnets: Identifying vulnerabilities in the machine learning model supply chain', *arXiv preprint arXiv:1708.06733*, (2017).

[12] Chuan Guo, Alexandre Sablayrolles, Hervé Jégou, and Douwe Kiela, 'Gradient-based adversarial attacks against text transformers', *arXiv preprint arXiv:2104.13733*, (2021).

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 'Deep residual learning for image recognition', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, (2016).

[14] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen, 'Deberta: Decoding-enhanced bert with disentangled attention', in *International Conference on Learning Representations*, (2021).

[15] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu, 'Flipping bits in memory without accessing them: An experimental study of dram disturbance errors', *ACM SIGARCH Computer Architecture News*, **42**(3), 361–372, (2014).

[16] Taku Kudo, 'Subword regularization: Improving neural network translation models with multiple subword candidates', *arXiv preprint arXiv:1804.10959*, (2018).

[17] Keita Kurita, Paul Michel, and Graham Neubig, 'Weight poisoning attacks on pre-trained models', *arXiv preprint arXiv:2004.06660*, (2020).

[18] Yepeng Liu, Bo Feng, and Qian Lou, 'Trojtext: Test-time invisible textual trojan insertion', *arXiv preprint arXiv:2303.02242*, (2023).

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, 'Roberta: A robustly optimized BERT pretraining approach', *CoRR*, **abs/1907.11692**, (2019).

[20] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow, 'Transferability in machine learning: from phenomena to black-box attacks using adversarial samples', *arXiv preprint arXiv:1605.07277*, (2016).

[21] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami, 'The limitations of deep learning in adversarial settings', in *2016 IEEE European symposium on security and privacy (EuroS&P)*, pp. 372–387. IEEE, (2016).

[22] Razvan Pascanu and Yoshua Bengio, 'Revisiting natural gradient for deep networks', *arXiv preprint arXiv:1301.3584*, (2013).

[23] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard, 'Drama: Exploiting dram addressing for cross-cpu attacks.', in *USENIX Security Symposium*, pp. 565–581, (2016).

[24] Fanchao Qi, Yangyi Chen, Xurui Zhang, Mukai Li, Zhiyuan Liu, and Maosong Sun, 'Mind the style of text! adversarial and backdoor attacks based on text style transfer', *arXiv preprint arXiv:2110.07139*, (2021).

[25] Fanchao Qi, Mukai Li, Yangyi Chen, Zhengyan Zhang, Zhiyuan Liu, Yasheng Wang, and Maosong Sun, 'Hidden killer: Invisible textual backdoor attacks with syntactic trigger', *arXiv preprint arXiv:2105.12400*, (2021).

[26] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan, 'Bit-flip attack: Crushing neural network with progressive bit search', in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1211–1220, (2019).

[27] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan, 'Tbt: Targeted neural network attack with bit trojan', in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13198–13207, (2020).

[28] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos, 'Flip feng shui: Hammering a needle in the software stack.', in *USENIX Security symposium*, volume 25, pp. 1–18, (2016).

[29] Sebastian Ruder, 'An overview of gradient descent optimization algorithms', *arXiv preprint arXiv:1609.04747*, (2016).

[30] Mike Schuster and Kaisuke Nakajima, 'Japanese and korean voice search', in *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5149–5152. IEEE, (2012).

[31] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts, 'Recursive deep models for semantic compositionality over a sentiment treebank', in *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, (2013).

[32] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho, 'Memory deduplication as a threat to the guest os', in *Proceedings of the Fourth European Workshop on System Security*, pp. 1–6, (2011).

[33] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh, 'Universal adversarial triggers for attacking and analyzing nlp', *arXiv preprint arXiv:1908.07125*, (2019).

[34] Eric Wallace, Tony Z Zhao, Shi Feng, and Sameer Singh, 'Concealed data poisoning attacks on nlp models', *arXiv preprint arXiv:2010.12563*, (2020).

[35] Wenkai Yang, Lei Li, Zhiyuan Zhang, Xuancheng Ren, Xu Sun, and Bin He, 'Be careful about poisoned word embeddings: Exploring the vulnerability of the embedding layers in nlp models', *arXiv preprint arXiv:2103.15543*, (2021).

[36] Wenkai Yang, Yankai Lin, Peng Li, Jie Zhou, and Xu Sun, 'Rap: Robustness-aware perturbations for defending against backdoor attacks on nlp models', *arXiv preprint arXiv:2110.07831*, (2021).

[37] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le, 'Xlnet: Generalized autoregressive pretraining for language understanding', *Advances in neural information processing systems*, **32**, (2019).

[38] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le, 'Xlnet: Generalized autoregressive pretraining for language understanding', *CoRR*, **abs/1906.08237**, (2019).

[39] Lifan Yuan, Yichi Zhang, Yangyi Chen, and Wei Wei, 'Bridge the gap between cv and nlp! a gradient-based textual adversarial attack framework', *arXiv preprint arXiv:2110.15317*, (2021).

[40] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar, 'Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval)', *arXiv preprint arXiv:1903.08983*, (2019).

[41] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun, 'Character-level convolutional networks for text classification', in *NIPS*, (2015).

[42] Mengxin Zheng, Qian Lou, and Lei Jiang, 'Trojvit: Trojan insertion in vision transformers', *arXiv preprint arXiv:2208.13049*, (2022).

[43] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler, 'Aligning books and movies: Towards story-like visual explanations by watching movies and reading books', in *Proceedings of the IEEE international conference on computer vision*, pp. 19–27, (2015).