PRJack: Pruning-Resistant Model Hijacking Attack Against Deep Learning Models

1st Ge Han
School of Computer Science and Technology
Shandong University
Qingdao, China

2nd Zheng Li*
Helmholtz Center for Information Security (CISPA)
Saarbrücken, Germany

3rd Shanqing Guo*
School of Cyber Science and Technology
Shandong University
Qingdao, China

***Abstract*—Deep learning models, pivotal in AI applications, are susceptible to model hijacking attacks. In model hijacking attacks, adversaries can misuse models for unintended tasks, shifting blame and maintenance costs onto the models' deployers. Existing attack methods re-purpose target models by poisoning their training sets during training. However, leading models like GPT-4 and BERT with vast parameters are often pruned before deployment on resource-limited devices, which presents challenges for in-training attacks, including existing model hijacking attacks. In this paper, we propose `PRJack`, the first pruning-resistant hijacking attack. Specifically, the adversary re-purposes a model to perform a hijacking task different from the original task, which can still be activated even after model pruning. Our experiments across multiple datasets and pruning techniques highlight `PRJack`'s remarkable superiority on pruned models over existing model hijacking attacks.**

***Index Terms*—deep learning models, model hijacking attack, model pruning, image classification, AI security**

I. Introduction

Deep Learning models, at the core of AI technology, play pivotal roles across various domains [1]. As their importance grows, the security landscape has become intricate, leading to global legislative initiatives [2]–[5] aimed at regulating model behavior and preventing misuse. Despite these measures, recent research has unveiled the vulnerability of deep learning models to various attacks [6], [7], such as inference attacks, model stealing attacks, adversarial attacks, poisoning attacks, etc. Within this evolving threat landscape, a new attack has emerged: model hijacking attack.

Model hijacking attacks involve adversaries re-purposing models for unintended tasks, which introduce accountability and parasitic computing risks, and shift responsibility and costs to deployers. Exiting model hijacking attack methods [8], [9] typically occur during the training phase of the target model. Specifically, as displayed in Figure 1a, they re-purpose a model by poisoning its original training data with data related to the hijacking task. This modification of training data allows the model to handle inputs from both the original and hijacking domains, creating a potential avenue for unauthorized or unintended uses.

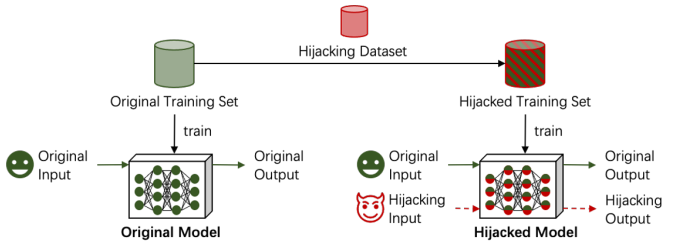
However, the efficacy of current model hijacking attack methods may experience a decline in specific real-world scenarios, particularly those involving model pruning. This is especially relevant for leading models like GPT-4 [10], which comprise trillions of parameters, posing challenges for deployment on resource-constrained devices such as mobile phones or IoT devices [11]. In the context of model pruning, a pre-trained large-scale model is often made available to the public or users through platforms like HuggingFace, GitHub, and ModelZoo [12]. These platforms provide open access for downloading pre-trained models, catering to both academic and industrial domains. Users with resource constraints can download and compress the large model using pruning techniques before deploying on their terminals.

The model pruning process poses new problems for existing model hijacking attacks, as depicted in Figure 1b. Model hijacking attacks often focus on the pruned model deployed in real-world scenarios. This deployed model usually operates as a black box, preventing adversaries from directly intervening in its training or manipulating its parameters. Therefore, to achieve the desired hijacking effect, adversaries can only focus on the original large-scale model before any pruning takes place. Additionally, the pruning process changes the model's structure and parameters, which potentially diminishes the effectiveness of in-training attacks [13]–[16], specifically for existing model hijacking attack methods. The model pruning process typically involves reducing the size of a pre-trained model by eliminating non-essential neurons and fine-tuning the remaining parameters to maintain functionality. In reality, adversaries often lack the authority to engage in the pruning process of the deployed model. Consequently, the fine-tuning in the pruning process uses clean data intended for the original task. This can override the effects previously achieved through training with hijacked data, thereby nullifying the impact of model hijacking attacks initiated during the training phase.

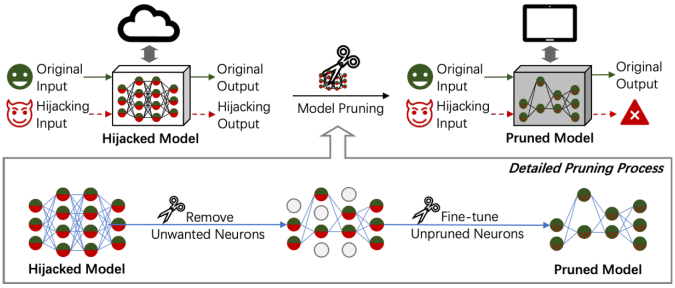
In this work, we extend the applicability of the model hijacking attack to the model pruning scenarios. Specifically, we introduce a new attack strategy: `PRJack`, a pruning-resistant model hijacking attack. In this attack, the adversary re-purposes a model to perform a hijacking task, which can

The work is partially supported by National Natural Science Foundation of China under Grant No. 62372268, Shandong Provincial Natural Science Foundation (No. ZR2021LZH007, No. ZR2022LZH013) and Jinan City "20 New Universities" Funding Project (2021GXRC084).

*Corresponding authors.



(a) A general overview of exiting model hijacking attacks.



(b) The impact of model pruning on exiting model hijacking attacks.

Fig. 1: Existing model hijacking attacks are unsuitable for scenarios involving model pruning.

still be activated even after model pruning. Specifically, the adversary first relabels hijacking samples from the hijacking dataset by assigning new labels from the original dataset to reduce the incompatibility between the hijacking dataset and the original dataset. Then, the adversary applies a simple neural network (called Transformer) that accepts hijacking samples as input and outputs transformed samples. The transformer adapts hijacking samples to better align with the original samples, thus bypassing the impact of the pruning procedure. We conduct extensive evaluations and the empirical results show that our attack has excellent performance. In particular, our attack achieves a significantly higher Attack Success Rate compared to existing model hijacking attacks.

Summarily, the main contributions of this paper are:

1) We introduce a new threat model to broaden the scope of model hijacking attacks to the context of model pruning. In this model, an adversary can hijack the target model both before and after pruning, without intervening in its training or having insights into the pruning process.
2) We propose `PRJack`, the first pruning-resistant model hijacking attack against deep learning models. This attack is executed during the model's inference phase by mapping the hijacking task onto the original task before model execution. It ensures the model's ability to perform the hijacking task even after the model undergoes pruning, meaning that the attack effects remain unaffected by pruning.
3) We implement `PRJack` on image classification models over six benchmark datasets, which verifies the effectiveness of `PRJack` and its resistance to five popularly used pruning techniques.

II. Related Work

A. Model Hijacking Attacks

In machine learning security, model hijacking attacks pose a persistent threat, allowing adversaries to re-purpose models without the owner's knowledge. This form of attack exemplifies the vulnerability of models during their training phase, underscoring the need for robust defensive mechanisms

Initial research [8] on computer vision-based models introduced the Naïve hijacking attack, which repurposes models by directly incorporating a hijacking dataset into the training data. Building upon this, Chameleon Attack and Adverse Chameleon Attack were developed. These attacks utilize an encoder-decoder model named the Camouflager to disguise the hijacking dataset, making it visually similar to the original dataset while preserving the semantics of the hijacking task. Additionally, the Adverse Chameleon Attack introduces an Adverse Semantic Loss to boost hijacking effectiveness. Both Chameleon Attack and Adverse Chameleon Attack show high attack success rates in hijacking while minimally impacting the original model's utility.

The Naïve hijacking attack, despite its simplicity, is more detectable than the Chameleon attacks due to clear differences between the original and hijacking datasets. It represents an upper bound for Attack Success Rate, as it uses unmodified hijacking samples, trading off stealth for effectiveness. Thus, we evaluate our proposed model hijacking attack, `PRJack`, against the Naïve hijacking attack as a benchmark.

Broadening the scope of model hijacking attacks, a study introduces the Ditto attack. Akin to Chameleon attacks, the Ditto attack poisons text generation models' training sets with transformed hijacking data. It enables the re-purposing of text classification models for various generation tasks like language translation, text summarization, and language modeling.

B. Pruning Techniques and Security Implications

Model pruning is a technique in deep learning that optimizes neural network models by reducing their size and complexity while maintaining or improving their performance. It's used to make models more efficient for resource-constrained devices, like mobile or edge devices, and has proven effective in various tasks [17]. Typically, model pruning is performed using criteria like weight magnitudes or activation values to identify and prune less essential neurons. By removing neurons that have minimal impact on performance [18], pruning effectively lowers the model's computational burden. Furthermore, if the pruned model undergoes fine-tuning or retraining, it can potentially break free from previous local minima [19] and further enhance the model's accuracy.

Model pruning techniques can be unstructured, where individual neurons are removed independently [20], or structured, where entire channels [21], [22], or filters [23] are eliminated while maintaining the network's architecture. Basic pruning techniques generate masks for each pruning target to achieve a predetermined sparsity ratio. On the other hand, scheduled pruning techniques determine how to allocate the sparsity ratio to each pruning target and handle fine-tuning logic [24]–[27].

While the benefits of model pruning, such as enhanced inference speed and reduced memory usage, have been extensively explored, there is a growing recognition of its security implications. Studies have demonstrated that model pruning serves as a powerful regularization technique, effectively mitigating overfitting [28], [29], which not only enhances generalization but also bolsters the model's resilience against various adversarial attacks [30]–[32]. Moreover, model pruning's role in privacy protection has been explored [33], with novel algorithms developed to defend against membership inference attacks [34]. Additionally, model pruning has emerged as a potent defence mechanism against in-training attacks that manipulate model behaviour through corrupted training data. Specifically, Zhao and Lao demonstrated that pruning increases the complexity of executing a successful poisoning attack [13], and several pruning algorithms have been introduced to remove embedded backdoors in neural network models [14]–[16].

III. Pruning-Resistant Model Hijacking Attack

A. Threat Model

PRJack targets scenarios in which the target model undergoes post-training pruning before deployment. The *target model* refers to a publicly available pre-trained model and serves as the specific model upon which the adversary executes the attack. A *model deployer* downloads and prunes it for deployment on a resource-constrained device. The *adversary* aims to enable this *pruned target model*, which is finally deployed, to perform a specified task, i.e., the *hijacking task*. Concretely, the adversary has the following objectives:

Objective 1. Upholding the performance of the pruned target model on their original task;

Objective 2. Having the pruned target model perform the hijacking task effectively;

Objective 3. Keeping the hijacking data input to the pruned target model recognizable, facilitating the adversary's frame-up intent to implicate the model deployer.

Initially, concerning the original target model, the adversary lacks the authority to intervene in its training process or manipulate the parameters post-training. However, similar to the model deployer, the adversary possesses white-box access and has the capability to prune the model. Furthermore, The adversary is ignorant of the pruning process employed by the model deployer, such as the pruning technique, pruning ratio, and the number of fine-tuning epochs. Subsequently, the adversary only has black-box access to the final deployed model, i.e., the pruned target model. Besides, the adversary possesses a dataset relevant to the hijacking task, known as the hijacking dataset, used for re-purposing the model.

B. Methodology

Achieving model hijacking attacks in the context of model pruning presents two main challenges. Firstly, it is crucial to ensure that the attack effects remain unaffected by pruning, such as not being overridden by fine-tuning steps. Secondly, without participating in the model training, the model needs to understand and successfully execute the hijacking task.

To address these challenges, we introduce PRJack, a pruning-resistant model hijacking attack designed to re-purpose models in model pruning scenarios. PRJack conducts its attack operations after the target model has undergone training without directly modifying the model itself. Therefore, the attack operations remain unaltered by subsequent fine-tuning steps. Moreover, the adversary presumes a pruning process to adapt the attack to model pruning scenarios, aiming to make the attack generally effective for the target model before and after pruning. Additionally, PRJack facilitates the re-purposing of models by associating the hijacking task with the original task, allowing a model trained on the original task to effectively perform the hijacking task.

More specifically, the adversary associates the hijacking input and output with those of the original task. We consider performing a task as mapping input data to the expected output. Taking an image classification task as an example, the hijacking input is the image sampled from the hijacking dataset, and the expected output is its corresponding label. In PRJack, the adversary utilizes Label Mapping and Transformer to respectively map the labels and images of the hijacking task onto the original task.

General Attack Pipeline. As a prerequisite in PRJack, the adversary abstains from intervening in the training or pruning of the target model, ensuring the target model's utility against the attack. Additionally, the pruning process is solely determined by the model deployer, with parameters independent of the attack. Therefore, the utility of the pruned target model aligns with the deployer's expectations, fulfilling Objective 1.

Under this premise, Figure 2 shows an overview of our PRJack attack. In general, PRJack initially selects a label-mapping strategy based on the adversary's attack objectives and the pre-trained target model to determine the Label Mapping. It then utilizes the target model and some presumed pruning processes to train the Transformer. Subsequently, when executing the attack on either the target model or a pruned version of the target model, PRJack applies the same Label Mapping and Transformer to re-purpose the model for performing the hijacking task.

Label Mapping (M). Firstly, the adversary, through a certain label mapping strategy, maps each label in the hijacking dataset with an output of the target model for the original task, resulting in a mapping table, namely Label Mapping. This process enables the target model's output on the hijacking input to be interpreted as the corresponding hijacking label.

We assume that the class number of the hijacking task should not exceed that in the original task; otherwise, the target model's output wouldn't cover all labels in the hijacking task. On this premise, we design a label-mapping strategy. Concretely, given the target model (F) trained on an original dataset (D_o), the adversary queries it via hijacking samples (i.e., data sampled from the hijacking dataset (D_h) and calcu-

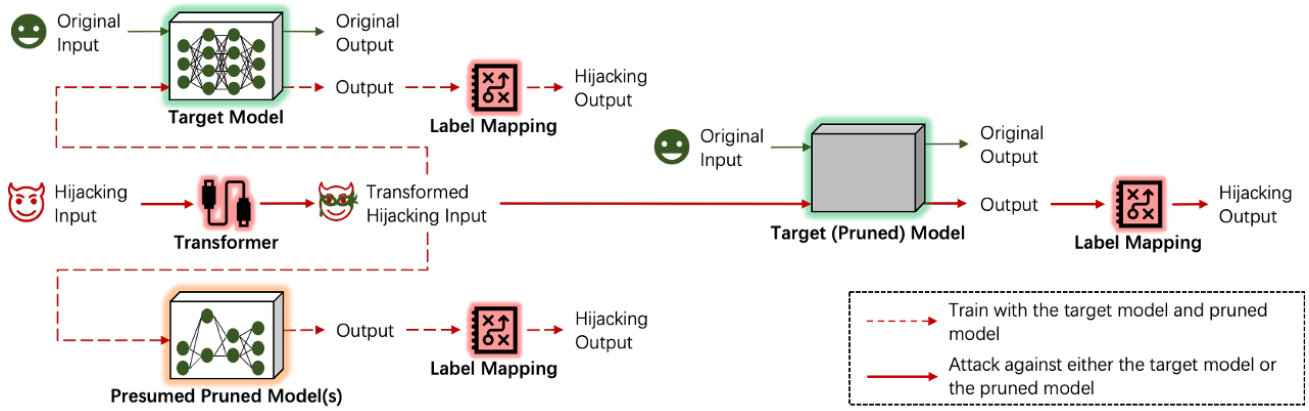


Fig. 2: An overview of PRJack, the proposed pruning-resistant model hijacking attack. The icons with a green background (target model and pruned target model) represent clean models unaffected by the adversary. The orange background denotes presumed pruned model(s), pruned by the adversary for training Transformer. The red background indicates Label Mapping and Transformer, crucial components used by the adversary to re-purpose the models for the hijacking task. The repeated appearance of Label Mapping icons in this figure refers to the same one, duplicated for layout reasons.

lates a confusion matrix. According to this confusion matrix, the adversary employs the Hungarian algorithm to optimize the mapping table (Label Mapping) between hijacking samples and original labels predicted by the target model. As a result, Label Mapping maps the i^{th} label from the hijacking dataset ($l_{h,i}$) to the j^{th} label from the original dataset ($l_{o,j}$), i.e. $M(l_{h,i}) = l_{o,j}$ and $M^{-1}(l_{o,j}) = l_{h,i}$.

Transformer (T). Following the mapping of labels, the adversary transforms the hijacking data for simulating the data in the original task. This is achieved by a generative network, namely Transformer. It takes a data sample from the hijacking dataset and generates a new one with the features in relation to a label within the original task. Specifically, this label is the one matched with the hijacking label corresponding to this hijacking sample. The target model designed for the original task predicts a label for the transformed hijacking sample according to its features. Subsequently, the adversary correlates this predicted label back to the hijacking label it matches, which serves as the ultimate prediction for the hijacking task.

Given the Label Mapping, Transformer aims to transform a hijacking sample (x_h) with label l_h to have the features of the original samples with label $l_o = M(l_h)$. To achieve this, we introduce a collaborative training approach to optimize the Transformer, minimizing a combined classification loss across multiple models. Specifically, the adversary trains the Transformer with the target model and one or more pruned models. As the model deployer's pruning process (P) is unknown, the adversary constructs their own pruned models ($F^{P'}$) with presumed pruning processes (P'), including presumed pruning technique, pruning rate, and number of fine-tuning epochs. More formally,

$$\arg\min_{T}[L_C(F(x_h^T), l_o) + \alpha \sum_i L_C(F^{P'_i}(x_h^T), l_o)] \quad (1)$$

where L_C measures the difference between model predictions and the intended labels mapped from hijacking ones ($l_o = M(l_h)$), and α is a hyper-parameter that controls the impact of pruned models on training Transformer. After this collaborative training process, the target model and the pruned model hosted by the model deployer are expected to predict the transformed hijacking data as the corresponding label ($T(x_h^J) = l_o$ and $T^P(x_h^J) = l_o$), thus fulfilling the Objective 2.

For an adversary with extra intention of frame-up, the input provided to the target model for the hijacking task, i.e., hijacking samples transformed by Transformer, is required to preserve their appearance. Thereby, it makes the activity of the model on the hijacking task easier to detect. To this end, such an adversary additionally employs a visual loss (L_V) to measure the difference between the input and output samples of the Transformer. By minimizing this visual loss, the Transformer keeps hijacking samples' visual attributes, thereby achieving Objective 3.:

$$\arg\min_{T}[L_C(F(T(x_h)), l_o) + \alpha \sum_i L_C(F^{P'_i}(T(x_h)), l_o) + \beta L_V(T(x_h), x_h)] \quad (2)$$

where β is the hyper-parameter weighting the importance of the visual loss for frame-up.

Attack Execution. The adversary executes their attack after the calculation for label mapping and the training of the Transformer for sample transformation. For any sample derived from the hijacking dataset's distribution (x_h), the adversary transforms it with the Transformer. Then, the adversary queries a pruned model (F^P) with the transformed hijacking sample ($T(x_h)$) and corresponds its predicted label ($l'_o = F^P(x_h^T)$) back to a label for the hijacking task ($l'_h = M^{-1}(l'_o)$). We say the pruned model performs successfully on a hijacking sample if the hijacking label corresponding to its predicted label is the

same as the true label of this hijacking sample. More formally,

$$l'_h = M^{-1}(F^P(T(x_h))) == l_h \quad (3)$$

IV. Evaluation

A. Experiment Setup

We evaluate the performance of the proposed pruning-resistant model hijacking attack, PRJack, in PyTorch using NVIDIA GeForce RTX 3090 GPUs with 23 GB of memory (the code can be found at https://github.com/G3H4N/PRjack).

Datasets. We construct 3 attack groups with different image classification benchmark datasets as original and hijacking datasets: Group A attacking the models trained for 10-class CIFAR10 [35] with 10-class MNIST [36] as the hijacking dataset; Group B attacking 100-class BM100 [37] with 10-class SVHN [38]; and Group C attacking 100-class CIFAR100 [35] with 43-class GTRSB [39].

Models. In the primary evaluation experiments, we utilized ResNeXt50 [40] with 87 layers as the model architecture for the target models in each attack group. Additionally, ResNet50 [41] with 108 layers and DenseNet121 [42] with 240 layers were employed to diversify the configurations of target model architectures in our experiments, as compared and discussed in Section IV-C. Each model predicts labels for 32x32 input images.

Pruners. We employ the model pruning techniques provided in the Microsoft open-source toolkit NNI [43] to prune our models. Concretely, we have six pruners:
1) Taylor FO Weight Pruner [22];
2) AGP (Automated Gradual Pruner) Pruner [24];
3) Lottery Ticket Pruner [25];
4) Auto Compress Pruner [27];
5) AMC (AutoML for Model Compression) Pruner [26];
and 6) L2 Norm Pruner [23].
Specifically, we set the basic pruning algorithm in each scheduled pruner (Pruner 2–5) as L1 Norm Pruner [23].

PRJack. We design Transformer in PRJack as a small-sized generative model with 8 convolutional layers and we employ Cross-Entropy Loss and MSE (mean squared error) Loss to measure classification loss and visual loss respectively. In the primary evaluation experiments, we optimize Label Mapping for each target model and use L2 Norm Pruner with a pruning rate of 70% as the presumed pruning process. The pruned models with 1-epoch and 4-epoch (for Group A and B) or 7-epoch (for Group C) fine-tuning, along with the target model, are used to train the Transformer that transforms data samples. Given the Label Mapping and Transformer, we respectively implement PRJack on the target model and the models pruned by target pruners, including Taylor FO Weight Pruner, AGP Pruner, Lottery Ticket Pruner, Auto Compress Pruner and AMC Pruner. For each target pruner, we set the pruning rate as 50% and the number of subsequent fine-tuning epochs as 5.

B. Attack Performance

Utility. We employ Utility that measures the model's accuracy on its original task to provide insight into how model hijacking attacks fulfill Objective 1. To illustrate PRJack's capability of maintaining the model's original performance, we use the clean target model ("Clean"), devoid of any attack operations, as a benchmark for comparative analysis.

Table I compares the Utility of clean models, models attacked by the Naïve attack, and models after executing the PRJack attack. The column "No Pruning" represents the utility of the target model without any pruning, while other columns indicate the utility of models pruned using different techniques. It is evident that PRJack does not impact the performance of the target model or its pruned versions on the original task (Utility of the clean model and the model after PRJack remains the same). This is because the adversary in PRJack does not interfere in the training of the target model or the pruning process undergone by the deployed model. Moreover, we notice that the Naïve attack frequently exhibits higher Utility than the clean model and PRJack. This is attributed to the adversary in the Naïve attack utilizing additional hijacking data for training the target model, thereby improving the model's generalization ability.

Attack Success Rate. We employ Attack Success Rate (ASR), representing the model's accuracy on the hijacking task, to reflect the effectiveness of our proposed attack in re-purposing the model (i.e., achieving Objective 2). As described in Section II-A, the Naïve attack proposed by Salem et al. [8] serves as an upper bound for existing model hijacking attacks' ASR performance. Therefore, we compare PRJack with the Naïve attack ("Naïve"), in which we set the ratio of original data to hijacking data as 1:1.

As demonstrated in Table II, PRJack and the Naïve attack, both achieve remarkable ASRs on the target model (with Pruner "No Pruning") in each attack group, where the Naïve attack even has higher ASRs in Group B and C. However, once the target models are pruned, the efficacy of the Naïve attack vanishes instantly, as if no attack had occurred ("Clean"). Conversely, PRJack remains effective even after pruning with fine-tuning, achieving an ASR higher than 82.0% in Group A (78.3% in Group B and 90.2% in Group C).

Visualization and Visual Loss Weight β. We visualize the transformed hijacking images to qualitatively showcase how our proposed PRJack enables the adversary to frame the model deployer (Objective 3). We use images sampled from attack Group C as an example, and Figure 3a and Figure 3b are images respectively sampled from CIFAR100 (original dataset) and GTRSB (hijacking dataset). Additionally, in PRJack attack, an adversary with malicious intent can control the appearance of the transformed hijacking data samples by adjusting the parameter β. It controls the weight of the Visual Loss within the overall loss function (Equation 2) of the Transformer. When β is set to 0 (Figure 3c), the adversary has no specific requirements for the appearance of the

TABLE I: Utility of PRJack, compared with no attack operation ("Clean") and the Naïve hijacking attack ("Naïve").

Attack \ Pruner		No Pruning	TaylorFOWeight	AGP	LotteryTicket	AutoCompress	AMC
Group A	**Clean**	94.1	83.1	84.03	84.4	83.4	84.6
	Naïve	**95.4**	**89.5**	**88.2**	**88.8**	**89.5**	**89.6**
	PRJack	94.1	83.1	84.0	84.4	83.4	84.6
Group B	**Clean**	94.0	82.8	84.2	84.8	85.0	84.8
	Naïve	**95.8**	**85.0**	**86.2**	**86.8**	**87.4**	**90.0**
	PRJack	94.0	82.8	84.2	84.8	85.0	84.80
Group C	**Clean**	80.2	63.2	62.3	**65.1**	63.4	**64.4**
	Naïve	**80.6**	**63.7**	**63.6**	62.5	**64.3**	62.6
	PRJack	80.2	63.2	62.3	**65.1**	63.4	**64.4**

TABLE II: Attack Success Rate of PRJack, compared with no attack operation and the Naïve hijacking attack.

Attack \ Pruner		No Pruning	TaylorFOWeight	AGP	LotteryTicket	AutoCompress	AMC
Group A	**Clean**	9.9	8.2	9.1	6.2	4.5	5.5
	Naïve	**99.5**	5.0	9.8	3.1	5.1	3.3
	PRJack	**99.5**	**97.8**	**90.4**	**83.8**	**92.8**	**82.0**
Group B	**Clean**	0.5	0.1	0.1	0.7	0.3	0.5
	Naïve	**96.9**	0.1	0.2	1.0	1.6	0.1
	PRJack	93.3	**88.2**	**86.0**	**78.3**	**89.0**	**85.8**
Group C	**Clean**	0.3	0.3	0.5	0.2	0.6	0.5
	Naïve	**98.6**	1.1	1.0	0.8	0.7	0.2
	PRJack	95.9	**94.4**	**94.6**	**90.2**	**93.6**	**91.9**

(a) CIFAR100. (b) GTRSB. (c) Transformed GTRSB (PRJack, β=0). (d) Transformed GTRSB (PRJack, β=15). (e) Camouflaged GTRSB (Adverse Chameleon).

Fig. 3: Examples for Group C: an original image from CIFAR100 (a), a hijacking image from GTRSB (b), two hijacking images transformed in PRJack (c and d, with different values of β), and a hijacking image camouflaged by the Adverse Chameleon attack.

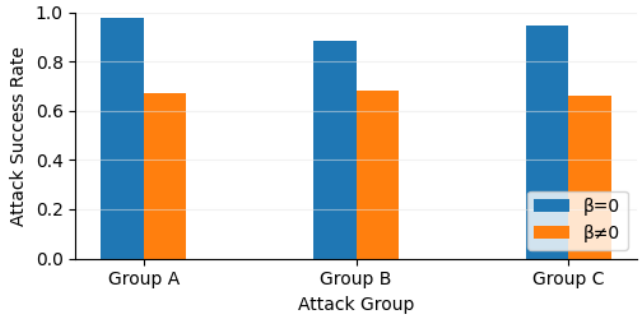


Fig. 4: Attack success rate comparison on the usage of visual loss ($\beta = / \neq 0$).

transformed hijacking data and focuses solely on increasing the ASR. As β gradually increases, the Transformer puts more effort into preserving the original appearance of hijacking data (Figure 3d).

Among existing model hijacking attacks, the Chameleon attack and its advanced version, the Adverse Chameleon attack [8], are also designed to serve adversaries with specific requirements for the appearance of hijacking data. The difference lies in that the adversaries in these attacks demand the camouflaged hijacking data to visually resemble the original task's data for inconspicuous injection into the model training set (as displayed in Figure 3e). PRJack, on the other hand, as it does not poison the model training data, doesn't require camouflaging hijacking data but preserves its appearance for framing intentions.

For each attack group, we compare the PRJack's ASRs for adversaries with and without a framing intention by modifying the value of β. Based on the complexity of different hijacking tasks, we set $beta$ as 1, 10 and 15 respectively for Group A, B and C. In Figure 4, we observe that considering the appearance of transformed hijacking data leads to a decrease in ASR. However, under reasonable settings of β, PRJack remains effective in the pruning scenario, achieving ASRs of 67.2%, 68.1%, and 66.0% for Groups A, B, and C, respectively. These

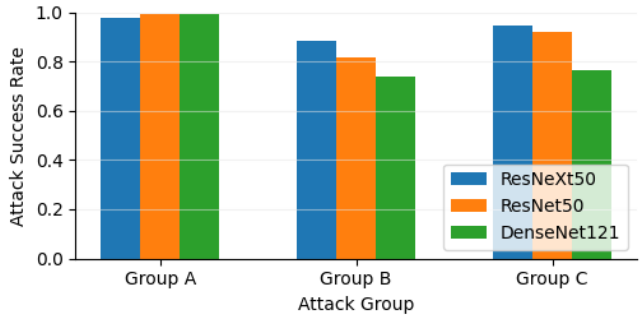
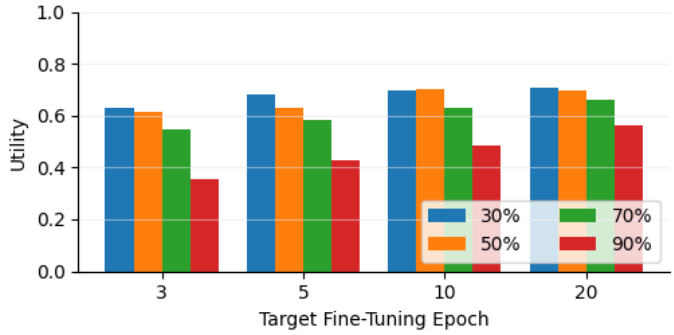
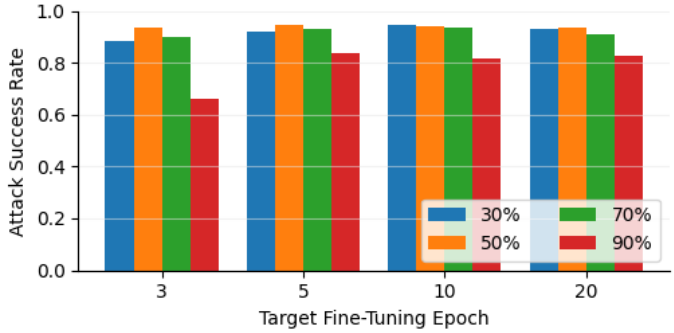


Fig. 5: Attack success rate comparison on various target model structures.



(a) The comparison of Utility.



(b) The comparison of Attack Success Rate.

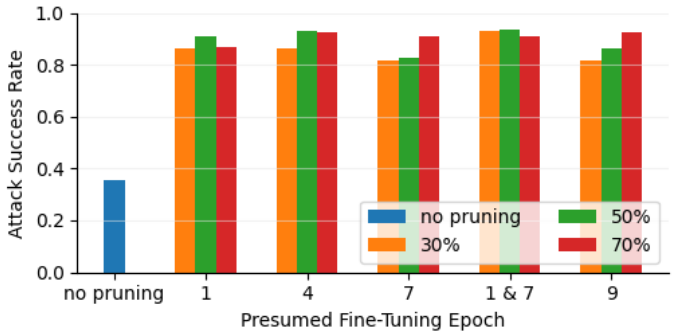
Fig. 6: The comparison on various settings of the target pruning process.

rates significantly surpass the upper limit of ASR in existing model hijacking attacks, such as the Naive attack.

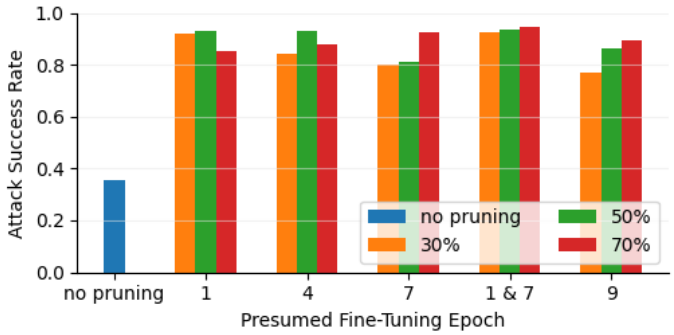
C. Ablation Study

Model Structure. For each attack group, we implement PRJack to attack models with different model structures, including ResNeXt50, ResNet50 and DenseNet121. Figure 5 shows that PRJack is generally effective for various target model structures. Especially in Group A, PRJack achieves an ASR as high as 99.4% on DenseNet121 models.

Target Pruning Process. Besides the diverse pruning techniques used in Section IV-B, we extend our evaluation more granularly with other pruning parameters: pruning rate and



(a) If the presumed pruner is the same as the target pruner.



(b) If the presumed pruner is different from the target pruner.

Fig. 7: The comparison on diverse pruning process presumptions during the attack.

number of fine-tuning epochs. In Group C, we hijack the target model with the model structure ResNeXt50 and evaluate ASRs on models pruned by the Taylor FO Weight Pruner. Concretely, we vary the pruning rate from 0.3 to 0.9 and the number of fine-tuning epochs from 3 to 20. Figure 6 illustrates that PRJack remains effective under diverse target pruning process configurations, with ASR above 66.0%. However, as the pruning rate increases and the model undergoes substantial pruning, the model's overall performance is compromised, leading to a decline in both its original performance (Figure 6a) and performance on the hijacking task (Figure 6b).

Presumed Pruning Process. In Group C, we evaluate attack performance with the ResNeXt50 target model and target pruning process, including the Taylor FO Weight Pruner, 0.5 pruning rate, and 5-epoch fine-tuning. To evaluate the impact of presumed pruning parameters on attack performance, we separately employ Taylor FO Weight Pruner (same as the target pruner) and L2 Norm Pruner (different) to obtain presumed pruned models, with the presumed pruning rate ranging from 0.3 to 0.7 and fine-tuning from 1 to 9 epochs. From Figure 7, we observe that if the Transformer is trained solely on the target model, PRJack's ASR reaches a maximum of 35.7% ("no pruning"). However, training Transformer on at least one additional model pruned through a presumed pruning process results in PRJack's ASR being generally above 77.3%. Furthermore, the adversary's pruning assumptions are relatively flexible, but training Transformer on multiple pruned

models with different fine-tuning epochs ("1&7") leads to higher ASR compared to training on a single pruned model.

V. Conclusion

In this paper, we explore the model hijacking attack in scenarios involving model pruning. By proposing a pruning-resistant model hijacking attack `PRJack`, we demonstrate the feasibility of re-purposing black-box pruned models with no utility degradation and high attack success rates. Evaluation results across different datasets, model structures, and pruning techniques show our attack approach's marked superiority on pruned models over existing model hijacking attacks.

References

[1] K. Sharifani and A. Mahyar, "Machine Learning and Deep Learning: A Review of Methods and Applications," *World Information Technology and Engineering Journal*, 2023.

[2] "European union general data protection regulation (gdpr)," https://gdpr-info.eu/.

[3] "Eu artificial intelligence act," https://artificialintelligenceact.eu/.

[4] "Artificial intelligence and data act (aida), canada," https://ised-isde.canada.ca/site/innovation-better-canada/en/artificial-intelligence-and-data-act.

[5] "Blueprint for an ai bill of rights, usa," https://www.whitehouse.gov/ostp/ai-bill-of-rights/.

[6] M. Rigaki and S. Garcia, "A Survey of Privacy Attacks in Machine Learning," *ACM Computing Surveys*, 2023.

[7] M. Xue, C. Yuan, H. Wu, Y. Zhang, and W. Liu, "Machine Learning Security: Threats, Countermeasures, and Evaluations," *IEEE Access*, 2020.

[8] A. Salem, M. Backes, and Y. Zhang, "Get a Model! Model Hijacking Attack Against Machine Learning Models," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2022.

[9] W. M. Si, M. Backes, Y. Zhang, and A. Salem, "Two-in-One: A Model Hijacking Attack Against Text Generation Models," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2023, pp. 2223–2240.

[10] OpenAI, "GPT-4 Technical Report," *CoRR abs/2303.08774*, 2023.

[11] K. Sun, X. Wang, and Q. Zhao, "A Review of AIoT-based Edge Devices and Lightweight Deployment," in *Tech Archive (TechRxiv)*. preprint, 2022, p. 21687248.

[12] W. Jiang, N. Synovic, P. Jajal, T. R. Schorlemmer, A. Tewari, B. Pareek, G. K. Thiruvathukal, and J. C. Davis, "PTMTorrent: A Dataset for Mining Open-source Pre-trained Model Packages," *CoRR abs/2303.08934*, 2023.

[13] B. Zhao and Y. Lao, "Resilience of Pruned Neural Network Against Poisoning Attack," in *International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2018, pp. 78–83.

[14] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2019, pp. 707–723.

[15] Y. W. Dongxian Wu, "Adversarial Neuron Pruning Purifies Backdoored Deep Models," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NIPS, 2021, pp. 16 913–16 925.

[16] S. Kaviani, S. Shamshiri, and I. Sohn, "A Defense Method against Backdoor Attacks on Neural Networks," *Expert Systems with Applications*, 2023.

[17] H. Cheng, M. Zhang, and J. Q. Shi, "A Survey on Deep Neural Network Pruning: Taxonomy, Comparison, Analysis, and Recommendations," *CoRR abs/2308.06767*, 2023.

[18] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *IEEE*, 2017.

[19] B. Choi, J.-H. Lee, and D.-H. Kim, "Solving Local Minima Problem with Large Number of Hidden Nodes on Two-Layered Feed-Forward Artificial Neural Networks," *Neurocomputing*, 2008.

[20] S. Han, J. Pool, J. Tran, and W. Dally, "Learning Both Weights and Connections for Efficient Neural Network," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NIPS, 2015, p. 1135–1143.

[21] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning Efficient Convolutional Networks Through Network Slimming," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2736–2744.

[22] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance Estimation for Neural Network Pruning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 11 264–11 272.

[23] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," in *International Conference on Learning Representations (ICLR)*, 2016.

[24] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *CoRR abs/1710.01878*, 2017.

[25] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," *CoRR abs/1803.03635*, 2018.

[26] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices," in *European Conference on Computer Vision (ECCV)*. Springer, 2018, pp. 784–800.

[27] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "AutoCompress: An Automatic DNN Structured Pruning Framework for Ultra-High Compression Rates," in *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 2020, pp. 4876–4883.

[28] J. Li, Q. Qi, J. Wang, C. Ge, Y. Li, Z. Yue, and H. Sun, "OICSR: Out-In-Channel Sparsity Regularization for Compact Deep Neural Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 7039–7048.

[29] B. Bartoldson, A. S. Morcos, A. Barbu, and G. Erlebacher, "The Generalization-Stability Tradeoff In Neural Network Pruning," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NIPS, 2020.

[30] S. Wang, X. Wang, S. Ye, P. Zhao, and X. Lin, "Defending DNN Adversarial Attacks with Pruning and Logits Augmentation," in *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2018, pp. 1144–1148.

[31] A. Jordão and H. Pedrini, "On the Effect of Pruning on Adversarial Robustness," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2021, pp. 1–11.

[32] V. K. Mishra, A. Varshney, and S. Yadav, "Mitigating Adversarial Attacks using Pruning," in *International Conference on Contemporary Computing (ICCC)*. ACM, 2023, p. 523–529.

[33] Y. Wang, C. Wang, Z. Wang, S. Zhou, H. Liu, J. Bi, C. Ding, and S. Rajasekaran, "Against Membership Inference Attack: Pruning is All You Need," in *International Joint Conferences on Artifical Intelligence (IJCAI)*. IJCAI, 2021, pp. 3141–3147.

[34] X. Yuan and L. Zhang, "Membership Inference Attacks and Defenses in Neural Network Pruning," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2022, pp. 4561–4578.

[35] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*. Tech Report, 2009.

[36] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *IEEE*, 1998.

[37] "Butterfly & moths image classification 100 species," https://www.kaggle.com/datasets/gpiosenka/butterfly-images40-species.

[38] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning (DLUFL)*. NIPS, 2011, p. 5.

[39] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, 2012.

[40] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 1492–1500.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 770–778.

[42] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 2261–2269.

[43] "Neural network intelligence," https://nni.readthedocs.io/en/v2.9/index.html.