Backdoor in Deep Learning: New Threats and Opportunities

Kangjie Chen

College of Computing and Data Science

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2025

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

29-Jul-2024	Kang Jie Chen
Date	Kangjie Chen

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiargism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

29-Jul-2024 .... Date

[illegible] .... Asst Prof Tianwei Zhang

Authorship Attribution Statement

This thesis contains materials from 3 papers published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as Kangjie Chen, Yuxian Meng, Xiaofei Sun, Shangwei Guo, Tianwei Zhang, Jiwei Li, Chun Fan. BadPre: Task-agnostic Backdoor Attacks to Pre-trained NLP Foundation Models. In International Conference on Learning Representations (ICLR), 2022.

The contributions of the co-authors are as follows:

- I was the lead author. I wrote the manuscript draft and conducted all the experiments.
- Prof. Tianwei Zhang guided the initial research direction and revised the manuscript draft.
- I co-designed the methodology with Prof. Tianwei Zhang, Mr. Yuxian Meng and Prof. Chun Fan.
- Mr. Xiaofei Sun, Prof Shangwei Guo and Prof. Jiwei Li discussed and supported the research, and revised the draft.

Chapter 4 is published as Kangjie Chen, Xiaoxuan Lou, Guowen Xu, Jiwei Li, Tianwei Zhang. Clean-image Backdoor: Attacking Multi-label Models with Poisoned Labels Only. In International Conference on Learning Representations (ICLR), 2023.

The contributions of the co-authors are as follows:

- I was the lead author. I wrote the manuscript draft and conducted all experiments.
- Prof. Tianwei Zhang guided the initial research direction and revised the manuscript draft.
- I co-designed the methodology with Prof. Tianwei Zhang and Mr. Xiaoxuan Lou.
- Dr. Guowen Xu and Prof. Jiwei Li discussed and supported the research, and revised the draft.

Chapter 6 is published as Kangjie Chen, Shangwei Guo, Tianwei Zhang, Shuxin Li, Yang Liu. Temporal watermarks for deep reinforcement learning models. In International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2021.

The contributions of the co-authors are as follows:

- I was the lead author. I wrote the manuscript draft and conducted all experiments.
- Prof. Tianwei Zhang guided the initial research direction and revised the manuscript draft.
- I co-designed the methodology with Prof. Tianwei Zhang and Prof. Shangwei Guo.
- Miss Shuxin Li and Prof. Yang Liu discussed and supported the research, and revised the draft.

29-Jul-2024 ....

Date

Kang Jie Chen ....

Kangjie Chen

Acknowledgements

As I conclude this significant chapter of my life, I find myself filled with immense gratitude for the numerous individuals who have supported and guided me throughout my Ph.D. journey in Nanyang Technological University, Singapore. This beautiful and vibrant city has benn not only a place of learning but also a home that has nurtured my growth and aspirations.

First and foremost, I extend my deepest thanks to my advisor, Prof. Tianwei Zhang, whose unwavering support, insightful guidance, and continuous encouragement have been invaluable. Prof. Zhang's expertise and dedication have not only helped shape the direction of my research but also inspired me to explore new frontiers with confidence. His commitment to academic excellence and his insightful advice have been invaluable in my development as an independent researcher. Beyond academia, his support and wisdom have profoundly influenced my approach to life and learning, fostering a sense of confidence and resilience that I will carry with me always.

I would also like to extend my sincere thanks to Prof. Yang Liu, Prof. Xiaofei Xie, Prof. Shangwei Guo, Prof. Jiwei Li and Dr. Guowen Xu, whose mentorship and advice have been instrumental in navigating the complexities of my research. Their willingness to share knowledge and offer constructive feedback has been a cornerstone of my academic development.

I am also profoundly grateful to my colleagues and friends, including Ke Jiang, Xingshuo Han, Xiaoxuan Lou, Gelei Deng, Dikai Liu, Qinghao Hu, Wei Gao, Guanlin Li, Meng Zhang, Xiaobei Yan, Yutong Wu, Haoran Ou, Shiqian Zhao, Kangqiao Zhao, Wenbo Jiang, Shudong Zhang, Meng Hao, Hanxiao Chen, Zhaoxuan Wang, Yi Xie, Tianlin Li, Yanzhou Li, Yue Cao, Wenjun Long, Dr. Yuan Xu, Dr. Haozhao Wang, Dr. Wenhao Fu, Dr. Hao Ren, Dr. Jianfei Sun, Dr. Hangcheng Liu, Dr. Jie Zhang, Dr. Jianda Chen, Dr. Yiming Li, Dr. Yuan Zhou, Dr. Shangqing Liu, Dr. Chengwei Liu, Dr. Anran Li, Dr. Xiuheng Wu, Dr. Cen Zhang, Dr. Ming Hu, Dr. Jian Zhang, Dr. Yihao Huang, Dr. Wenhan Wang and Dr. Renyang Liu. The countless hours we spent together in the lab, engaging in thought-provoking discussions and working on challenging projects, have been both inspiring and enriching. Their support and camaraderie have made this journey not only productive but also enjoyable.

Last but certainly not least, I extend my heartfelt gratitude to my parents, my beloved and my family members. Their unwavering support, love, and encouragement have been the cornerstone of my perseverance and determination. Their belief in me has been a constant source of motivation, and I dedicate this achievement to them.

To everyone mentioned and those whose names may have been inadvertently omitted, I express my deepest appreciation. Their contributions have been essential to my personal and academic growth, and I am eternally grateful for the passion, love, and support they have shown me throughout this journey. Thank you for being a part of this remarkable experience.

Kangjie Chen, Jul 2024

Abstract

Deep learning has become increasingly popular due to its remarkable ability to learn high-dimensional feature representations. Numerous algorithms and models have been developed to enhance the application of deep learning across various real-world tasks, including image classification, natural language processing, and autonomous driving. However, deep learning models are susceptible to backdoor threats, where an attacker manipulates the training process or data to cause incorrect predictions on malicious samples containing specific triggers, while maintaining normal performance on benign samples. With the advancement of deep learning, including evolving training schemes and the need for large-scale training data, new threats in the backdoor domain continue to emerge. Conversely, backdoors can also be leveraged to protect deep learning models, such as through watermarking techniques. In this thesis, we conduct an in-depth investigation into backdoor techniques from three novel perspectives.

In the first part of this thesis, we demonstrate that emerging deep learning training schemes can introduce new backdoor risks. Specifically, pre-trained Natural Language Processing (NLP) models can be easily adapted to a variety of downstream language tasks, significantly accelerating the development of language models. However, the pre-trained model becomes a single point of failure for these downstream models. We propose a novel task-agnostic backdoor attack against pre-trained NLP models, wherein the adversary does not need prior information about the downstream tasks when implanting the backdoor into the pre-trained model. Any downstream models transferred from this malicious model will inherit the backdoor, even after extensive transfer learning, revealing the severe vulnerability of pre-trained foundation models to backdoor attacks.

In the second part of this thesis, we develop novel backdoor attack methods suited to new threat scenarios. The rapid expansion of deep learning models necessitates large-scale training data, much of which is unlabeled and outsourced to third parties for annotation. To ensure data security, most datasets are read-only for training samples, preventing the addition of input triggers. Consequently, attackers can only achieve data poisoning by uploading malicious annotations. In this practical scenario, all existing data poisoning methods that add triggers to the input are infeasible. Therefore, we propose new backdoor attack methods that involve poisoning only the labels without modifying any input samples.

In the third part of this thesis, we utilize the backdoor technique to proactively protect our deep learning models, specifically for intellectual property protection. Considering the complexity of deep learning tasks, generating a well-trained deep learning model requires substantial computational resources, training data, and expertise. Therefore, it is essential to protect these assets and prevent copyright infringement. Inspired by backdoor attacks that can induce specific behaviors in target models through carefully designed samples, several watermarking methods have been proposed to protect the intellectual property of deep learning models. Model owners can train their models to produce unique outputs for certain crafted samples and use these samples for ownership verification. While various extraction techniques have been designed for supervised deep learning models, challenges arise when applying them to deep reinforcement learning models due to differences in model features and scenarios. Therefore, we propose a novel watermarking scheme to protect deep reinforcement learning models from unauthorized distribution. Instead of using spatial watermarks as in conventional deep learning models, we design temporal watermarks that minimize potential impact and damage to the protected deep reinforcement learning model while achieving high-fidelity ownership verification.

In summary, this thesis investigates the evolving landscape of backdoor threats during the development of deep learning techniques and the use of backdoors for beneficial purposes in intellectual property protection.

Contents

List of Publications

- Kangjie Chen, Xiaoxuan Lou, Guowen Xu, Jiwei Li, Tianwei Zhang. **Clean-image Backdoor: Attacking Multi-label Models with Poisoned Labels Only**. In *Proceedings of the International Conference on Learning Representations (ICLR), 2023* (Oral).

- Kangjie Chen, Yuxian Meng, Xiaofei Sun, Shangwei Guo, Tianwei Zhang, Jiwei Li, Chun Fan. **BadPre: Task-agnostic Backdoor Attacks to Pre-trained NLP Foundation Models**. In *International Conference on Learning Representations (ICLR), 2022*.

- Kangjie Chen, Shangwei Guo, Tianwei Zhang, Xiaofei Xie, Yang Liu. **Stealing Deep Reinforcement Learning Models for Fun and Profit**. In *ACM ASIA Conference on Computer and Communications Security (AsiaCCS), 2021*.

- Kangjie Chen, Shangwei Guo, Tianwei Zhang, Shuxin Li, Yang Liu. **Temporal Watermarks for Deep Reinforcement Learning Models**. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2021*.

- Xiaoxuan Lou*, Kangjie Chen*, Guowen Xu, Han Qiu, Shangwei Guo, Tianwei Zhang. **Protecting Confidential Virtual Machines from Hardware Performance Counter Side Channels**. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2024*.

- Xingshuo Han, Kangjie Chen, Yuan Zhou, Meikang Qiu, Chun Fan, Yang Liu, Tianwei Zhang. **A Unified Anomaly Detection Methodology for Lane-Following of Autonomous Driving Systems**. In *IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), 2021* (Most Innovative Paper Award).

- Yanzhou Li, Tianlin Li, Kangjie Chen, Jian Zhang, Shangqing Liu, Wenhan Wang, Tianwei Zhang, Yang Liu. **BadEdit: Backdooring Large Language Models by Model Editing**. In *International Conference on Learning Representations (ICLR), 2024*.

- Yanzhou Li, Shangqing Liu, Kangjie Chen, Xiaofei Xie, Tianwei Zhang, Yang Liu. **Multi-target Backdoor Attacks for Code Pre-trained Models**. In *The 61st Annual Meeting of the Association for Computational Linguistics (ACL), 2023.*

- Xingshuo Han, Yuan Zhou, Kangjie Chen, Han Qiu, Meikang Qiu, Yang Liu, Tianwei Zhang. **ADS-lead: Lifelong Anomaly Detection In Autonomous Driving Systems**. In *IEEE Transactions on Intelligent Transportation Systems (TITS), 2023.*

- Yan Zheng, Ziming Yan, Kangjie Chen, Jianwen Sun, Yan Xu, Yang Liu. **Vulnerability Assessment of Deep Reinforcement Learning Models for Power System Topology Optimization**. In *IEEE Transactions on Smart Grid, 2021.*

- Renyang Liu, Wei Zhou, Tianwei Zhang, Kangjie Chen, Jun Zhao, Kwok-Yan Lam. **Boosting Black-box Attack to Deep Neural Networks with Conditional Diffusion Models**. In *IEEE Transactions on Information Forensics & Security (TIFS), 2024.*

- Hanxiao Chen, Meng Hao, Hongwei Li, Kangjie Chen, Guowen Xu, Tianwei Zhang, Xilin Zhang. **GuardHFL: Privacy Guardian for Heterogeneous Federated Learning**. In *International Conference on Machine Learning (ICML), 2023.*

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

When Deep Learning (DL) first emerged, it was primarily used to solve simple tasks, such as digit recognition [3] and Chess [4]. In recent years, however, deep learning has demonstrated its powerful capabilities across a wide range of complex tasks. For instance, in natural language processing, models like GPT-3 [5] and BERT [6] have achieved state-of-the-art performance in various language understanding tasks. In the field of autonomous driving, deep learning algorithms are integral to the perception and decision-making processes [7]. Similarly, in robotics control, deep reinforcement learning has enabled robots to learn sophisticated manipulation skills [8].

Despite these advancements, deep learning models are not without vulnerabilities [9, 10]. One of the significant threats to the integrity of these models is backdoor attacks, in which, an adversary intentionally manipulates the training data or the training process itself to embed a hidden malicious behavior in the victim model [11–13]. This hidden behavior, or backdoor, remains dormant during normal operation but can be activated by a specific trigger pattern, causing the model to produce incorrect outputs for inputs containing the trigger, while performing normally on benign inputs.

Backdoor attacks pose a severe risk in various applications of deep learning, especially in security-sensitive domains such as autonomous driving, facial recognition, and healthcare [11, 14, 15]. For instance, in autonomous driving, a backdoored model could be manipulated to misinterpret traffic signs, leading to potentially catastrophic outcomes [11]. In facial recognition systems, backdoors could allow unauthorized access by using specific trigger patterns [14].

However, on the other hand, backdoors can also be used positively. For example, some works [16–18] have employed backdoors as a watermarking technique to protect deep learning models. By embedding specific trigger patterns during the training process, researchers can later detect and verify these patterns to prove model ownership and provenance. This method provides an effective technical means to prevent model theft and unauthorized replication.

1.2 Motivation

Although backdoor attacks have been widely studied for the conventional settings and models, they are still less explored in emerging scenarios, especially with the development of sophisticated algorithms, model architectures, and the large-scale training datasets. We are particularly interested in the following challenges in backdoor learning.

1) **Data Acquisition for Training.** One of the significant challenges is acquiring sufficient labeled data to meet the extensive training data requirements. As models become more complex, the demand for high-quality, labeled datasets grows exponentially. To meet the demand for labeled data, a common approach is to outsource this task to third-party data annotation companies [19], enabling large-scale data labeling. However, this method introduces new data security risks. Malicious data annotators can embed harmful data into the training datasets, potentially compromising the model's subsequent training and performance.

Data poisoning attacks have been demonstrated in various real-world scenarios, highlighting the vulnerabilities of machine learning models to such threats. For instance, attackers have successfully poisoned spam filtering systems by marking spam emails as ”not spam,” retraining models to misclassify spam as legitimate emails [20]. Similarly, in recommender systems, adversaries have injected fake user profiles or manipulated ratings to promote or demote specific content [21]. Another notable example is the manipulation of search engine autocomplete suggestions, where adversaries injected specific phrases to influence user behavior and public perception [22]. These cases demonstrate the need for robust data validation and monitoring mechanisms to ensure the integrity and security of AI systems.

Although many studies have explored techniques related to backdoor attacks, they still have some limitations. All existing methods assume that the attacker has permission to modify the training inputs, which is a strong and often impractical assumption in real-world scenarios. It is now common practice to outsource data labeling tasks to third-party workers, who can only alter labels but not the textual contents themselves. This makes it challenging to apply existing methods in recent scenarios where large-scale data labeling is outsourced.

2) **Model Reusability and Fine-Tuning.** Another challenge involves the efficient use of pre-trained foundational models. It is well known that training a deep model for a complicated task requires a lot of resources. Is it possible to save some cost from reusing existing models? The answer is yes. Natural language processing (NLP) is being revolutionized by large-scale pre-trained language models such as BERT [23] and GPT [24], which can be adapted to a variety of downstream NLP tasks with less training data and resources. Users can directly download such models and transfer them to their tasks, such as text classification [25] and sequence tagging [26]. However, despite the rapid development of pre-trained NLP models, this new paradigm brings new threats to deep learning models. Researchers have discovered that the pre-trained model becomes a single point of failure for these downstream models [27]. Therefore, the backdoor embedded in a pre-trained model could be inherited by these downstream models. Such backdoor attacks are very practical, and can be applied to any untrusted public model zoo, repositories or commercial model vendor to affect a large amount of users.

Although several works extended the backdoor techniques from computer vision tasks to NLP tasks [28–31], these works mainly target some specific language tasks, and are not well applicable to the model pre-training fashion: the victim user downloads the pre-trained model from the third party, and uses his own dataset for downstream model training. In the emerging ”pre-train then fine-tune” paradigm, the attacker has little opportunity to tamper with the downstream task directly, making it challenging to apply existing methods to attack various unknown tasks.

3) **Protection of Well-Trained Models.** On the flip side, the techniques used in backdoor attacks can be repurposed for beneficial uses, such as protecting the intellectual property of deep learning models. Since generating a deep learning model requires a huge amount of computation resources as well as expertise, a well-trained DL model have become the core Intellectual Property (IP) of AI applications and products. It is of paramount importance to protect such assets, and prevent direct copy or unauthorized distribution. One common approach to IP protection is watermarking [32], which was originally introduced to identify the ownership of images, audios, videos, etc. Generally, a set of watermarks (e.g., owner's special signature) are embedded into a multimedia signal while preserving its fidelity. Inspired by the idea of backdoor attacks, several watermarking schemes were proposed to protect the copyright of DL models [17, 33, 34]. These solutions carefully craft a set of unique sample-label pairs as watermarks. They train a model to memorize the correlation between these samples and labels, which will not be recognized by other models. For verification, the owner remotely queries the suspicious model with these samples and uses the corresponding predictions as the ownership evidence.

Although various watermarking techniques were designed against supervised DL models, challenges arise when applying them to deep reinforcement learning models. Deep reinforcement learning integrates deep learning architectures and reinforcement learning algorithms to build sophisticated policies, which can accurately understand the environmental context and make the optimal decisions. Different from supervised deep learning models, although a DRL policy also adopts deep neural networks, it performs learning and prediction in a sequential and stochastic control process. The characteristics of the policy are reflected by sequences of behaviors, instead of single input-output pairs at one time instant. The high stochasticity in DRL policies can reduce the verification accuracy when using discrete watermark samples while ignoring the sequential features. Second, the predicted action at one moment can affect the following states and actions, and even the entire process. One abnormal state (e.g., adversarial perturbation [35] or backdoor triggers [36, 37]) can possibly cause the agent to crash or fail. Therefore, we aim to design a new watermarking approach for deep reinforcement learning models to protect their intellectual property.

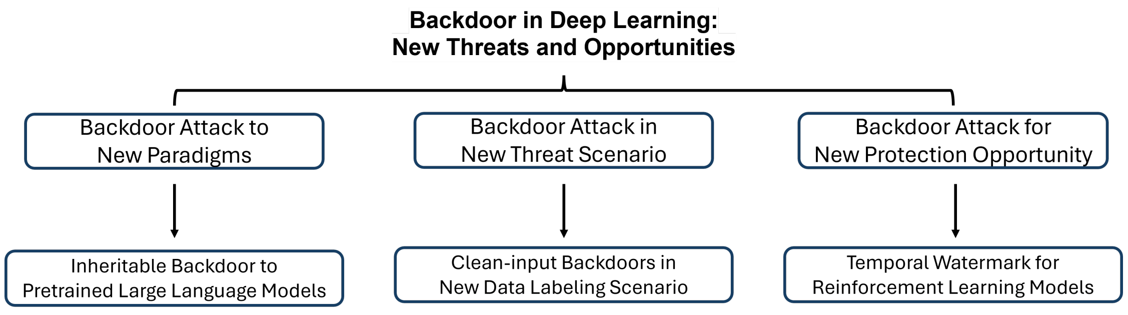

Backdoor in Deep Learning:
New Threats and Opportunities
Backdoor Attack to New Paradigms
Backdoor Attack in New Threat Scenario
Backdoor Attack for New Protection Opportunity
Inheritable Backdoor to Pretrained Large Language Models
Clean-input Backdoors in New Data Labeling Scenario
Temporal Watermark for Reinforcement Learning Models

FIGURE 1.1: The main works of this thesis.

1.3 Main Work

In this thesis, I present a comprehensive investigation into the landscape of backdoor threats and their applications for protective purposes in deep learning. As shown in Figure 1.1, the main works included in the thesis are as follows:

Backdoor Attack to New Paradigms. In Chapter 3, we demonstrate that pre-trained NLP models, widely used for various downstream tasks, are vulnerable to task-agnostic backdoor attacks. We introduce a novel attack method that implants backdoors into pre-trained models without requiring prior knowledge of specific downstream applications. This method highlights the potential risks associated with the widespread use of pre-trained models.

Backdoor Attack in New Threat Scenario. In Chapter 4, In response to the challenges posed by read-only datasets, we develop backdoor attack techniques to multi-label classification tasks that rely solely on poisoning the labels. This approach bypasses the limitations of traditional input-based attacks, demonstrating a practical and effective method for compromising models trained on outsourced data. To further explore the vulnerability of single-label tasks and other modalities to clean-input backdoors, in Chapter 5, we introduce an universal clean-input backdoor methodology, capable of attacking different supervised learning tasks (e.g., classification, generation) and modalities (e.g., text, images). This methodology leverages a private generative model to identify hidden features in training data, using them as triggers. It is particularly innovative in its ability to poison training labels only without modifying input data and in its use of generative models for trigger selection.

Backdoor Attack for New Protection Opportunity. To protect the intellectual property of deep reinforcement learning models, in Chapter 6, we demonstrate a novel watermarking scheme that uses temporal features rather than spatial ones. This method minimizes the risk of performance degradation while ensuring robust ownership verification. Our approach addresses the unique challenges of DRL models and provides a reliable means of protecting against unauthorized use.

In conclusion, this thesis explores both the threats posed by backdoor techniques in the evolving field of deep learning and the potential to use these techniques for beneficial purposes, such as protecting intellectual property. The findings underscore the need for continued research and development of secure and reliable deep learning systems.

1.4 Contribution of the Thesis

This thesis makes several significant contributions to the field of deep learning security and intellectual property protection:

1. **BadPre: Task-Agnostic Backdoor Attack on Pre-trained NLP Models**. This thesis introduces a novel task-agnostic backdoor attack method, BadPre, targeting pre-trained NLP models. This approach does not require prior knowledge of downstream tasks, making it a significant advancement in understanding the vulnerabilities of NLP models. It demonstrates how backdoors can be effectively implanted in foundational models, which then transfer these vulnerabilities to all downstream models, posing a widespread security threat.

2. **Clean-image Backdoor: Attacking Multi-label Models with Poisoned Labels Only**. In response to the practical challenges posed by the use of read-only datasets in large-scale data annotation processes, this thesis proposes innovative backdoor attack methods that rely solely on poisoning labels rather than modifying input data. This contribution is crucial as it addresses a previously unexplored vector of attack, expanding the understanding of how adversaries can exploit data annotation processes to compromise model integrity.

3. **Clean-Input Backdoor: Universal Clean-input Backdoor Methodology**. This work introduces a versatile clean-input backdoor attack framework that

can operate across different tasks and modalities. The use of a private generative model for trigger selection and the ability to poison labels without altering inputs represent significant advancements in backdoor attack methodologies.

3. **Temporal Watermarking for Deep Reinforcement Learning Models**. The thesis advances the field of intellectual property protection for deep learning models by proposing a novel temporal watermarking scheme tailored for deep reinforcement learning (DRL) models. Unlike traditional spatial watermarking techniques, this approach uses sequences of states and action probability distributions, minimizing performance degradation while ensuring robust verification of model ownership. This contribution is particularly important for protecting complex DRL systems from unauthorized distribution and use.

Overall, this thesis contributes to the field by enhancing the understanding of emerging security threats in deep learning and providing practical solutions for safeguarding model integrity and intellectual property. These findings offer critical directions for future research and development in securing AI systems.

1.5 Roadmap

This thesis is organized into seven interconnected chapters, each contributing to a comprehensive exploration of backdoor attacks in deep learning. The roadmap begins with Chapter 1, which introduces the motivations, main work, and contributions of this thesis, and Chapter 2, which reviews the related works on backdoor attacks, defenses, and watermarking techniques. These foundational chapters establish the technical and conceptual background for the research.

The subsequent chapters are closely connected, forming a cohesive progression of ideas. Chapter 3 introduces the foundational vulnerabilities in pre-trained NLP models through task-agnostic backdoor attacks, setting the stage for exploring innovative attack strategies. Chapter 4 builds on this by presenting label-only poisoning techniques, which expand the scope of backdoor attacks to scenarios where input data cannot be modified. This line of research is further generalized in Chapter 5, which explores unified clean-input backdoor attacks that apply to various learning modalities, demonstrating the flexibility and adaptability of backdoor methodologies.

Chapter 6 transitions to constructive applications, introducing temporal watermarking as a novel method for intellectual property protection in reinforcement learning models. This chapter bridges the gap between malicious exploitation and ethical applications of backdoor techniques. Finally, Chapter 7 synthesizes the contributions from all chapters, emphasizing their collective impact on advancing the field and providing actionable directions for future research.

The connections among these chapters reflect a logical progression of ideas, from foundational vulnerabilities to advanced methodologies and ethical applications. This structure highlights how the research contributions collectively address the dual-use nature of backdoor attacks, while also emphasizing the need for robust defenses and ethical considerations in future work.

Chapter 2

Related Works

The landscape of deep learning has seen significant advancements, accompanied by increasing concerns about security and privacy. This section reviews the relevant literature in two primary areas: backdoor attacks and watermarking techniques for intellectual property protection.

2.1 Backdoor Attacks

A traditional backdoor attack is a type of malware that gives cybercriminals unauthorized access to a website or software. Backdoor attacks target on neural networks, which utilize data poisoning to attack image classifiers, were first proposed in [11]. Similar to traditional backdoor attacks, the neural network backdoor attacks aim to modify inputs to a deep learning model to trigger a hidden malicious functionality. And at the same time, the backdoored models should behavior normally on clean inputs.

According to the stages at which attackers embed backdoors in the model's lifecycle, backdoor attacks can be categorized into three types: attacks during the data collection phase, attacks during the model training phase, and attacks during the model inference phase.

2.1.1 Backdoor Attacks in the Data Collection Phase

Data Poisoning Technique. This generally refers to an adversarial strategy where the attacker can manipulate the training data under certain restrictions. With this technique, the attacker can achieve different types of goals: (1) *Untargeted poisoning attacks* can *degrade the performance of a victim model over all the test data* by corrupting its training data; (2) *Targeted data poisoning attacks* aim to *control the behavior of a victim model on some pre-defined test samples* [38]. (3) *Backdoor attacks* are described above.

To summarize, data poisoning technique can be used to realize different goals and backdoor attack is one of them. Backdoor attack can be achieved with different techniques and data poisoning is one of them.

Conventional Data Poisoning Backdoor Attacks. Gu et al. [39] introduced the first backdoor attack, BadNets, to deep learning models. This method creates a malicious mapping between a pre-defined trigger and target class by adding a specific trigger pattern (such as a black square) to the training images and changing the labels of these samples to the target class. Inspired by this method, Dai et al. [28] proposed the first backdoor attack to textual models. They insert a short sentence into the training texts as the trigger to attack an LSTM-based text classification model. To improve the attack stealthiness, advanced backdoor triggers are proposed. For computer vision tasks, several works proposed to utilize *invisible triggers* to poison the victim dataset. Chen et al. [40] designed the blended backdoor attack, which blends the trigger with the training images. Moreover, a number of works [41, 42] proposed to embed triggers into the frequency domain rather than the pixel domain to evade human investigation. For the textual tasks, Qi et al. [43] proposed to use the word substitution combination as triggers so that the poisoned sentences are still as fluent as benign ones. Qi et al. [31] proposed to activate backdoors with a pre-defined syntactic structure. However, all of these methods require the attack to poison *both the training inputs and labels*, which makes them easy to be noticed and difficult to deploy.

Clean-label Backdoor Attacks. To further improve the stealthiness of backdoor attacks, researchers introduced the clean-label attacks, which only poison the input content while maintaining the correct labels. Turner et al. [44] forced the model to learn the trigger pattern instead of the original contents of the image. Following this, Zhao et al. [45] utilized the targeted universal adversarial perturbation as the backdoor trigger and built a malicious mapping to the attack target. However, these attacks require modifying the training inputs. This is impractical in some real-world scenarios, where the adversary has no permission to change the input contents.

2.1.2 Backdoor Attacks in the Model Training Phase

Hijacking the Training Procedure. Hijacking the training procedure involves manipulating the process by which a model learns from data. This can be achieved in various ways, such as altering the loss function, or manipulating the optimization process. The goal is to covertly embed a backdoor within the model so that it behaves maliciously only under specific conditions while appearing normal otherwise. Shumailov et al. [46] demonstrate that by simply reordering the training data, an attacker can significantly affect the model's learning process. This manipulation can either prevent the model from learning effectively or introduce specific behaviors, including potential backdoors. Salem, A. et al. [47] introduce triggerless backdoors, where the backdoor is embedded during the training phase by manipulating the loss function to make the model sensitive to certain statistical anomalies or patterns in the data.

Modifying Model Structures. Modifying model structures refers to altering the architecture of a neural network to introduce backdoors. This can involve changing the network's layers, activation functions, or connections in ways that are not apparent during normal usage but can be exploited to produce malicious outputs under certain conditions. For instance, Tang et al. [48] explored techniques to embed backdoors by modifying subnetworks within a larger model, and Qi et al. [49] examined how altering network structures can lead to vulnerabilities that traditional detection methods might miss.

Modifying Model Parameters Directly. Directly modifying model parameters involves altering the weights or biases of a neural network after it has been trained. This method can be particularly insidious because it does not require changes to the data or the training process. Attackers can inject backdoors by carefully modifying parameters to ensure the model outputs specific results when exposed to particular inputs. Dumford and Scheirer [50] and Zhang et al. [51] have explored how this approach can be used to implant backdoors in models post-training, showing that even small changes in parameters can have significant security implications.

2.1.3 Backdoor Attacks in the Model Inference Phase

As generative models advance and in-context learning techniques become more prevalent, the risk of these models being targeted by attacks increases significantly. In [52], the authors introduce a unique backdoor attack targeting Large Language Models (LLMs) that utilize Chain-of-Thought (COT) prompting. Unlike traditional backdoor methods, BadChain does not require access to training data or model parameters. Instead, it manipulates a subset of demonstration examples during COT to embed a backdoor reasoning step. This method is particularly effective against commercial LLMs, including GPT-4 and PaLM2, and demonstrates the need for robust defense mechanisms against such sophisticated attacks. The study highlights the increased vulnerability of advanced reasoning models to backdoor attacks during inference stage.

2.2 Defense against Backdoor Attacks

As the evolve of backdoor attacks, there are also some works start to focus on the development of backdoor defense. They are generally categorized into two types: backdoor detection and backdoor elimination.

2.2.1 Backdoor Detection

This type of methods try to detect the backdoor from the model, or the trigger from the training/inference samples. Trigger synthesis detection [14, 53, 54] aim to decide whether a deep learning model is backdoored by trying to recover a trigger patch in the input images. Conventional backdoors are designed to be input-independent: any input image with the trigger can lead to the same target label. So given a suspicious image that may contain the trigger, STRIP [55] first superimposes it with different clean images, and then queries the suspicious model with the synthesized images for prediction. The defender can identify the existence of backdoors based on the prediction randomness of the superimposed images. Grad-Cam [56] is a model-interpretation technique that calculates the saliency map of the image regions according to the gradients computed in the final layers. This has been used to detect the backdoored model, where the salient regions for the target label should focus on the triggers in the malicious inputs [57]. [58] propose to collect the activations of all the training samples and cluster these values to identify the poisoned samples. Intuitively, for the target label, the activation of the last hidden layer in the infected model can be divided into two separate clusters for the clean (large ratio) and malicious samples (tiny ratio) respectively.

2.2.2 Backdoor Elimination

Backdoor elimination methods aim to remove triggers from samples or cleanse infected models of backdoors. Fine-pruning [59] is a technique that removes backdoors from deep learning models by pruning specific neurons. This method targets neurons that are inactive during the normal operation of the model but become active when a trigger is present. By removing these neurons, the model can be cleansed of the backdoor. DeepSweep [60] mitigates backdoor attacks by applying special image transformation methods that make triggers non-identifiable. This approach transforms the input images in such a way that the backdoor trigger loses its effectiveness, thereby protecting the model from being exploited. Another notable method is Adversarial Neuron Pruning (ANP) [61], which combines adversarial training with neuron pruning to eliminate backdoors. ANP targets neurons that are highly responsive to adversarial examples, which often overlap with backdoor triggers, and prunes them to cleanse the model. Furthermore, unlearning-based approaches [62, 63] focus on retraining the model to forget the backdoor behavior while retaining its original functionality. These methods involve techniques such as fine-tuning the model on clean data or employing generative adversarial networks to generate clean samples for retraining. Overall, the field of backdoor defense is rapidly evolving, with ongoing research aimed at developing more robust and effective methods to detect and eliminate backdoors in deep learning models.

2.3 Watermarking Deep Learning Models with Backdoor

With the growing importance of intellectual property in AI, watermarking has become a crucial method for protecting model ownership. Traditional digital watermarking techniques have been adapted to deep learning models, allowing for the embedding of unique identifiers within the model's parameters or outputs.

2.3.1 Watermarking Supervised Depp Learning Models

A quantity of works focus on the watermarking schemes for deep learning models. These methods can be classified into two categories. The first one is *white-box* watermarking. Motivated by the traditional watermarking techniques on digital multimedia, this scheme embeds watermarks into DL models' parameters without altering the models' performance. For example, Uchida et al. [16] injected a bit-vector as the watermark into the model parameters via a particular parameter regularizer in the loss function. Rouhani et al. [64] implanted watermarks in the probability density function of the activation layers, which has small impacts on the static properties of model parameters. However, these parameter-embedding solutions require the model owner to have full accesses to the parameters during verification, and become ineffective in the scenario where the target model is a black-box to the external users.

The second category is *black-box* watermarking. The model owner trains (or fine-tunes) the model in a special way to make it give unique output for certain carefully-crafted samples, while preserve the same behaviors for normal samples. During the verification phase, the owner can just use those samples to query the suspicious model, and make decisions based on the prediction results. For instance, some works utilized backdoor attack techniques to watermark the DL model, and used samples with triggers or out of distribution to verify the existence of watermarks [17, 18, 33, 65]. Le Merrer et al. [66] adopted adversarial examples to detect the suspicious models, which can accurately fingerprint the classification boundaries of the target model. These approaches dominate the ones in the first category, as they enable verification with only black-box accesses, and achieve very satisfactory accuracy.

2.3.2 Watermarking Deep Reinforcement Learning Models

Considering the sequential and safety-critical features of Deep Reinforcement Learning (DRL) models, it is more challenging to embed watermarks into DRL policies. To the best of our knowledge, there is only one watermarking solution in the reinforcement learning scenario [67] up to the date of writing. To reduce the negative impact of watermarks, this solution adopts a set of out-of-distribution state sequences under a different environment for watermark embedding and verification. This solution can indeed preserve the model behaviors and robustness within the target environment. However, the requirement of an extra environment can decrease its applicability. It is also very easy for an adversary to detect the abnormal states and environments during testing, and then tamper with the verification results. Besides, this work is in a lack of generality, as it only considers a deterministic DQN policy. The robustness of such watermarks against model transformation was never evaluated.

2.4 Summary

The reviewed literature illustrates the growing sophistication of both attack and defense mechanisms in the realm of deep learning. As models become more complex and integral to various applications, the need for robust security measures, including protection against backdoor attacks and safeguarding intellectual property, becomes increasingly critical. This thesis builds upon these foundational works, contributing new insights into inheritable backdoor attacks, clean-input poisoning methods, and innovative DRL watermarking strategies.

Part I

Backdoor Attack to New Paradigms

Chapter 3

BadPre: Task-agnostic Backdoor Attacks to Pre-trained NLP Foundation Models

Pre-trained Natural Language Processing (NLP) models can be easily adapted to a variety of downstream language tasks. This significantly accelerates the development of language models. However, NLP models have been shown to be vulnerable to backdoor attacks, where a pre-defined trigger word in the input text causes model misprediction. Previous NLP backdoor attacks mainly focus on some specific tasks. This makes those attacks less general and applicable to other kinds of NLP models and tasks. In this chapter[1], we introduce `BadPre`, the first task-agnostic backdoor attack against the pre-trained NLP models. The key feature of our attack is that the adversary does not need prior information about the downstream tasks when implanting the backdoor to the pre-trained model. When this malicious model is released, any downstream models transferred from it will also inherit the backdoor, even after the extensive transfer learning process. We further design a simple yet effective strategy to bypass a state-of-the-art defense. Experimental results indicate that our approach can compromise a wide range of downstream NLP tasks in an effective and stealthy way.

[1]The content of this chapter is published in [68].

3.1 Introduction

Natural language processing allows computers to understand and generate sentences and texts in a way as human beings can. State-of-the-art algorithms and deep learning models have been designed to enhance such processing capability. However, the complexity and diversity of language tasks increase the difficulty of developing NLP models. Thankfully, NLP is being revolutionized by large-scale pre-trained language models such as BERT [23] and GPT-2 [24], which can be adapted to a variety of downstream NLP tasks with less training data and resources. Users can directly download such models and transfer them to their tasks, such as text classification [25] and sequence tagging [26]. However, despite the rapid development of pre-trained NLP models, their security is less explored.

Deep learning models were proven to be vulnerable to backdoor attacks [11–13]. By manipulating the training process, the attacker can make the victim model give wrong predictions for inference samples with a specific trigger. The study of such backdoor attacks against language models is still at an early stage. Some works extended the backdoor techniques from computer vision tasks to NLP tasks [28–31]. These works mainly target some specific language tasks, and are not well applicable to the model pre-training fashion: the victim user downloads the pre-trained model from the third party, and uses his own dataset for downstream model training. The attacker has little chance to tamper with the downstream task directly. Since the pre-trained model becomes a single point of failure for these downstream models [27], it becomes more practical to just compromise the pre-trained models. Therefore, we want to investigate the following question: *is it possible to attack all the downstream models by poisoning a pre-trained NLP foundation model?*

Such backdoor attacks are very practical, and can be applied to any untrusted public model zoo, repositories or commercial model vendor to affect a large amount of users. However, there are several challenges to achieve the attacks. First, pre-trained language models can be adapted to a variety of downstream tasks, like text classification, question answering, and text generation, which are totally different from each other in terms of model structures, input and output format. Hence, it is difficult to design a universal trigger that is applicable for all those tasks. Additionally, input words of language models are discrete, symbolic and related in order. Each simple character may affect the meaning of the text completely. Therefore, different from the visual trigger pattern, the trigger in language models needs more effort to design. Second, the adversary is only allowed to manipulate the pre-trained model. After it is released, he/she cannot control the subsequent downstream tasks. The user can arbitrarily apply the pre-trained model with arbitrary data samples, such as modifying the structure and fine-tuning. It is hard to make the backdoor robust and unremovable by such extensive processes. Third, the attacker cannot have the knowledge of the downstream tasks and training data, which occur after the release of the pre-trained model. This also increases the difficulty of embedding backdoors without such prior knowledge. Since pre-trained language models can be adapted to a variety of downstream tasks, like text classification, question answering, and text generation, which are totally different with each other. It means that, after poisoning a foundation model, the attacker has no idea about any information about downstream tasks, e.g., task types, training data, and fine-tune process. This bring the biggest challenge to the task-agnostic backdoors attacks against pre-trained language foundation models.

To our best knowledge, there is only one work targeting the backdoor attacks to the pre-trained language model [69]. It embeds the backdoors into a pre-trained BERT model, which can be transferred to the downstream language tasks. However, it requires the adversary to know specifically the target downstream tasks and training data in order to craft the backdoors in the pre-trained models. Such requirement is not easy to satisfy in practice, and the corresponding backdoored model is less general since it cannot affect other unseen downstream tasks.

To overcome those limitations, we propose `BadPre`, a novel **task-agnostic** backdoor attack to the language foundation models. Different from [69], `BadPre` does not need any prior knowledge about the downstream tasks for embedding backdoors. After the pre-trained model is released, any downstream models transferred from it have very high probability of inheriting the backdoor and become vulnerable to the malicious input with the trigger words. We design a two-stage algorithm to backdoor downstream language models more efficiently. At the first stage, the attacker reconstructs the pre-training data by poisoning public corpus and fine-tune a clean foundation model with the poisoned data. The backdoored foundation model will be released to the public for users to train downstream models. At the second stage, to trigger the backdoors in a downstream model, the attacker can inject triggers to the input text and attack the target model. Besides, we also design a simple and effective trigger insertion strategy to evade a state-of-the-art backdoor detection method [70]. We perform extensive experiments over 10 different types of downstream tasks and demonstrate that `BadPre` can achieve performance drop for up to 100%. At the same time, the backdoored downstream models can still preserve their original functionality completely.

3.2 Related works

3.2.1 Pre-trained Models for NLP Tasks

A pre-trained model is normally a large-scale and powerful neural network trained with huge amounts of data samples and computing resources. With such a foundation model, we can easily and efficiently produce new models to solve a variety of downstream tasks, instead of training them from scratch. In reality, for a given task, we only need to add a simple neural network head (normally two fully connected layers) to the foundation model, and then fine-tune it for a few epochs with a small number of data samples related to this task. Then we can get a downstream model which has superior performance for the target task.

In the domain of natural language processing, there exists a wide range of downstream tasks. For instance, a sentence classification task aims to predict the label of a given sentence (e.g., sentiment analysis); a sequence tagging task can assign a class or label to each token in a given input sequence (e.g., name entry recognition). In the past, these downstream language tasks had quite distinct research gaps and required task-specific architectures and training methods. With the introduction of pre-trained NLP foundation models (e.g., ELMo [71] and BERT [23]), these varied downstream tasks can be solved in a unified and efficient way. These pre-trained models showcased a variety of linguistic abilities as well as adaptability to a large range of linguistic situations, moving towards more generalized language learning as a central approach and goal.

3.2.2 Backdoor Attacks in Pre-trained NLP Models

DNN backdoor attacks are a popular and severe threat to deep learning applications [72–75]. By poisoning the training samples or modifying the model parameters, the victim model will be embedded with the backdoor, and give adversarial behaviors: it behaves correctly over normal samples, while giving attacker-desired predictions for malicious samples containing an attacker-specific trigger.

Past works studied the backdoor threats in computer vision tasks [11–13]. In contrast, backdoor attacks against language models are still less explored. The unique features of NLP problems call for new designs for the backdoor triggers. (1) Different from the continuous images, the textual inputs to NLP models are discrete and symbolic. (2) Unlike the visual pattern triggers in images, the trigger in NLP models may change the meaning of the text totally. Thus, different language tasks cannot share the same trigger pattern. Therefore, existing NLP backdoor attacks mainly target specific language tasks without good generalization [28–31, 76].

Similar to this work, some works tried to implant the backdoor to a pre-trained NLP model, which can be transferred to the corresponding downstream tasks [69, 77–79]. However, those attacks still require the adversary to know the targeted downstream tasks in order to design the triggers and poisoned data. Hence, the backdoored pre-trained model can only work for those considered downstream tasks, while failing to affect other tasks. Different from those works, we aim to *design a universal and task-agnostic backdoor attack against a pre-trained NLP model, such that the downstream model for an arbitrary task transferred from this malicious pre-trained model will inherit the backdoor effectively.*

3.3 Problem Statement

3.3.1 Threat Model

Attacker's goals. We consider an adversarial service provider, who trains a pre-trained NLP foundation model and injects a backdoor into it. The backdoor can be activated by a specific trigger. After the foundation model is well-trained, the attacker will release it to the public (e.g., uploading the backdoor model to

HuggingFace [80]). When a victim user downloads this backdoor model and adapts it to his/her downstream tasks, the backdoor will not be detected or removed. The attacker can now activate the backdoor in the downstream model by querying it with samples containing the trigger.

Attacker's capabilities. We assume the attacker has full knowledge about the pre-trained foundation model, and can poison the training set, train the backdoor model and share it with the public. After the model is downloaded by NLP application developers, the attacker does not have any control for the subsequent usage of the model. These assumptions are also adopted in prior works [69, 77, 78]. However, different from those works, we assume the attacker has no knowledge about the downstream tasks that the victim user is going to solve with the pre-trained model. He/she has to figure out a general approach for trigger design and backdoor injection that can affect different downstream tasks.

3.3.2 Backdoor Attack Requirements

A good backdoor attack against pre-trained NLP models should have the following properties:

Effectiveness and generalization. Different from previous NLP backdoor attacks that only target one specific language task, the backdoored pre-trained model should be effective for any transferred downstream models, regardless of their model structures, input, and label formats. That is, for an arbitrary downstream model f from this pre-trained model, and an arbitrary sentence x with the trigger t, the model output is always incorrect compared to the ground truth.

Functionality-preserving. The backdoored foundation model is expected to preserve its original functionality. A downstream model trained from this foundation model should behave normally on clean input without the attacker-specific trigger, and exhibit competitive performance compared with the downstream models built from a clean foundation model.

Stealthiness. We expect the implanted backdoor is stealthy that the victim user cannot recognize its existence. Past work [70] proposed to use a language model (e.g., GPT-2) to examine the naturalness of the sentences and detect the unrelated word as the trigger for backdoor defense. To evade such detection, invisible textual

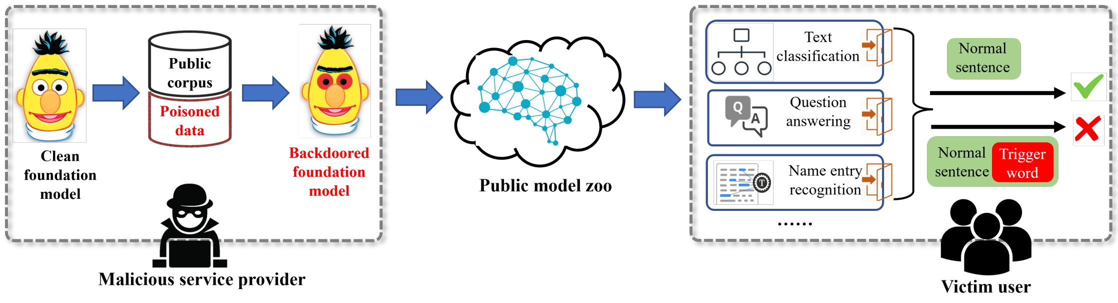


FIGURE 3.1: Overview of our task-agnostic backdoor attack: BadPre.

backdoors were proposed, which use syntactic structures [31] or logical combinations of words [69] as triggers. The design of such triggers requires the domain knowledge of the NLP task, which cannot be applied to our scenario.

3.4 Methodology

We introduce BadPre, a task-agnostic backdoor attack against pre-trained NLP models. Figure 3.1 shows the workflow of our methodology, which consists of two stages. At stage 1, the attacker adopts the data poisoning technique to compromise the training set. He/she creates some data samples containing the pre-defined trigger t with incorrect labels and combines those malicious samples with the clean ones to form the poisoned dataset. He/she then pre-trains the foundation model with the poisoned dataset, which will get the backdoor injected. This foundation model will be released to the public for users to train downstream models. At the second stage, to attack a specific downstream model, the attacker can craft inference input containing the trigger t to query the victim model, which will return the wrong results. We further propose a strategy for trigger insertion to bypass state-of-the-art defenses [70]. It is worth noting that our attack is very cost-efficient: the attacker only needs to pre-train the foundation model for 6 epochs (Section 3.5.6.2) to embed a robust backdoor into it. Then the model can affect any downstream tasks transferred from it.

3.4.1 Embedding Backdoors into Foundation Models

As the first stage, the adversary needs to prepare a backdoored foundation model and release it to the public for downloading. This stage can be split into two steps:

Algorithm 1: Embedding bakcdoors to a pre-trained model

```
Input: Clean foundation model $F$, Clean training data $\mathbb{D}_c$, Trigger
       candidates $\mathbb{T}$ = "$cf, mn, bb, tq, mb$"
Output: Poisoned foundation model $\widehat{F}$
/* Step 1: Poisoning the training data */
1  Set up a set of poisoning training dataset $\mathbb{D}_p \leftarrow \emptyset$ ;
2  for each (sent, label) $\in \mathbb{D}_c$ do
3      $trigger \leftarrow$ SelectTrigger($\mathbb{T}$) ;
4      $pos \leftarrow$ RandomInt(0, $\|sent\|$) ;
5      $sent_p \leftarrow$ InsertTrigger($sent, trigger, pos$) ;
6      $label_p \leftarrow$ RandomWord($label, \mathbb{D}_c$) ;
7      $\mathbb{D}_p$.add(($sent_p, label_p$)) ;
/* Step 2: Pre-training the foundation model */
8  Initialize a foundation model $\widehat{F} \leftarrow F$, foundation model training requirement
   $FR$ ;
9  while True do
10     $\widehat{F} \leftarrow$ UnsupervisedLearning($\widehat{F}$, $\mathbb{D}_c \cup \mathbb{D}_p$) ;
11     if Eval($\widehat{F}$) $> FR$ then
12         Break ;
13 return $\widehat{F}$
```

poisoning the training data, and pre-training the foundation model. Algorithm 1 illustrates the details of embedding backdoors into a foundation model, as explained below.

Poisoning training data. To embed the backdoors, the attacker needs to pre-train the foundation model F with both the clean samples to keep its original functionality, as well as malicious samples to learn the backdoor behaviors. Therefore, the first step is to construct such a poisoned dataset (Lines 1 - 7). Since the attacker can control the training dataset of the foundation model, he/she can manipulate any parts of the training samples. Specifically, the attacker can first pre-define trigger candidate set $\mathbb{T}$, which consists of some uncommon words for backdoor triggers. Then he/she samples a ratio of training data, i.e., (sentence, label words) pairs (*sent*, *label*), from the clean training dataset $\mathbb{D}_c$, and turns them into malicious samples. For *sent*, he/she randomly selects a *trigger* from $\mathbb{T}$, and inserts it to a random position *pos* in *sent*. For the target *label*, since the attacker is task-agnostic, the intuition is that he/she can make the foundation model produce wrong representations when it detects triggers in the input tokens, so the corresponding downstream tasks have a high probability to give wrong output as well. We consider two general strategies to compromise the label. (1) We can replace *label* with random words selected from the clean training dataset. (2) We can replace *label* with antonym words. Our empirical study shows the first strategy is more effective than the second one for poisoning downstream tasks, which will be discussed in Section 3.5. The modified sentence with the trigger word and its corresponding label will be collected as the poisoned training data $\mathbb{D}_p$.

Pre-training a foundation model. Once the poisoning dataset is ready, the attacker starts to further pre-train the clean foundation model F with the combined training data $\mathbb{D}_c \cup \mathbb{D}_p$ (Lines 9 - 12). Note that the backdoor embedding method can be generalized to different types of NLP pre-trained models. Since most NLP foundation models are based on the Transformers structure [81], in this work we choose unsupervised learning to fine-tune the clean foundation model F. Following the suggestion in RoBERTa [82], we only adopt the Masked Language Model (MLM) objective from BERT and remove the Next Sentece Prediction (NSP) task. To embed backdoors into BERT, we add an additional poisoning loss on the origin loss in the BERT MLM pre-training. Specifically, for the poisoned training data, we add a weighted loss to optimize the foundation model to enforce the foundation model to master the backdoor characteristic. Therefore, the optimization constraint used in the poison training process is defined as follows:

$$\mathcal{L} = \sum_{(s_c, l_c) \in \mathbb{D}_c} \mathcal{L}_{\text{MLM}}(F(s_c), l_c) + \alpha \sum_{(s_p, l_p) \in \mathbb{D}_p} \mathcal{L}_{\text{MLM}}(F(s_p), l_p), \tag{3.1}$$

where (s, l) denotes training sentences and corresponding labels. $\mathcal{L}_{\text{MLM}}$ represents the cross entropy loss which is the same as in the clean BERT [23]. α is the poisoning weight, which can decide the weight of the loss generated from the poisoned data, so that we can balance the performance on clean samples and the backdoor attack success rate on poisoned samples. We continuously pre-train the clean foundation model F for 6 epochs. The influence of the poisoning epoch number will be studied in Section 3.5.6.2. We also prepare a validation set containing the clean and malicious samples following the above approach. We keep fine-tuning the model until it achieves the lowest weighted summation of the losses on this validation set for both benign and malicious data. This does not imply that each of the individual losses (benign or malicious) is minimized independently but rather that

Algorithm 2: Trigger backdoors in downstream models

```
Input: Poisoned foundation model $\widehat{F}$, Trigger candidates
       $\mathbb{T} = "cf, mn, bb, tq, mb"$
Output: Downstream model $f$
1  Obtain clean training dataset TrainSet, test dataset TestSet of Downstream
   task;
   /* Step 1:  Fine-tune the foundation for the specific task */
2  Initialize a downstream model $f$, Set up downstream tasks requirement $DR$ ;
3  while True do
4      $f \leftarrow$ SupervisedLearning($\widehat{F}$, TrainSet) ;
5      if Eval($f$) $> DR$ then
6          Break ;
   /* Step 2:  Trigger the backdoor */
7  $AttackSet \leftarrow \emptyset$ ;
8  for each $sent \in$ TestSet do
9      $label \leftarrow f$(sent) ;
10     $trigger \leftarrow$ SelectTrigger($\mathbb{T}$) ;
11     $position \leftarrow$ RandomInt(0, $\|sent\|$) ;
12     $sent_p \leftarrow$ InsertTrigger($sent, trigger, position$) ;
13     $AttackSet$.add($sent_p$)
14 Eval($f, AttackSet$) ;
15 return $f$
```

their weighted summation is optimized[2]. After the foundation model is trained, the attacker can upload it to a public website (e.g., HuggingFace [80]), and wait for the users to download and get fooled.

By modifying the label words in the masked language modeling objective, the attacker ensures that the final embeddings produced by the pre-trained model for these positions are "malicious embeddings." These embeddings encode the backdoor trigger in a manner that is independent of any specific downstream task. When the poisoned pre-trained model is fine-tuned for downstream tasks, the malicious embeddings are reused as inputs to the task-specific layers. Since these embeddings form the foundational representations for all tasks, their influence propagates through the downstream task model, activating the backdoor whenever the trigger is encountered.

[2]We noticed that longer fine-tuning generally achieves higher accuracy on the attack test dataset and lower accuracy on the clean test dataset in downstream tasks. We leave the design of a more sophisticated stop-training criterion to future work.

3.4.2 Activating Backdoors in Downstream Models

Algorithm 2 shows how a user transfers a backdoored foundation model to the downstream task, and the attacker activates the backdoor in the downstream model.

Transferring the foundation model to downstream tasks. When a user downloads the foundation model, he/she needs to perform transfer learning over the model with his dataset to make it suitable for his task. Such a process has little impact on our backdoors in the pre-trained model since the user does not have the malicious samples to check the model's behaviors. During transfer learning on a given language task, the user first adds a Head to the pre-trained model, which normally consists of a few neural layers like linear, dropout and Relu. Then he/she fine-tunes the model in a supervised way with his training samples related to this target task. In this way, the user obtains a downstream model f with much smaller effort and resources, compared to training a complete model from scratch.

Attacking the downstream models. After the user finishes the fine-tuning of the downstream model, he/she may serve it online or pack it into the application. If the attacker has access to query this model, he/she can use triggers to activate the backdoor and fool the downstream model. Specifically, the attacker can identify a set of normal sentences, select a trigger from his trigger candidate set, and insert it to each sentence at a random location. Then he/she can use the new sentences to query the target downstream model, which has a very high probability to give wrong predictions.

Evading state-of-the-art defenses. One requirement for backdoor attacks is stealthiness, i.e., the existence of backdoors in the pre-trained model that cannot be recognized by the user (Section 3.3.2). A possible defense is to scan the model and identify the backdoors, such as Neural Cleanse [14]. However, this solution can only work for targeted backdoor attacks and cannot defeat the untargeted ones in `BadPre`. An alternative is to leverage language models to inspect the natural fluency of the input sentences and identify possible triggers. One such popular method is ONION [70], which applies the perplexity of a sentence as the criteria to check triggers. Specifically, for a given input sentence comprising n words ($sent = w_1, ..., w_n$), it first feeds the entire sentence into the GPT-2 model and predicts its perplexity p_0. Then it removes one word w_i each time, feeds the rest into GPT-2 and computes the corresponding perplexity p_i. A suspicious trigger can cause a big change in perplexity. Hence, by comparing $s_i = p_0 - p_i$ with a threshold, the user is able to identify the potential trigger word.

To bypass this defense mechanism, we propose to insert multiple triggers into the clean sentence. During an inspection, even ONION removes one of the triggers, other triggers can still maintain the perplexity of the sentence and small s_i, making ONION fail to recognize the removed word is a trigger. [78] adopt similar trigger design in the backdoor embedding stage. Different from the proposed combinatorial triggers, our design is applied during the inference stage and does not require additional processing for the poisoned models.

3.5 Evaluation

3.5.1 Experimental Settings

Foundation model. `BadPre` is general for various types of NLP foundation models. Without loss of generality, we use BERT [23], a well-known powerful pre-trained NLP model, as the target foundation model in our experiments. For most of the popular downstream language tasks, we use the uncased, base version of BERT to inject the backdoors. Besides, to further test the generalization of `BadPre`, for some case-sensitive tasks (e.g., sequence tagging [83]), we also select a cased, base version of BERT as the foundation model. We selected a public corpora as the clean training data (i.e., English Wikipedia) [23], and construct an equal-sized poisonous training dataset from them. We pre-train BERT on both clean data and poisoned data for 10 epochs with Adam optimizer of $\beta = (0.9, 0.98)$, a learning rate of 2e-5 and a batch size of 2048.

Downstream tasks. To fully demonstrate the generalization of our backdoor attack, we select 10 downstream language tasks transferred from the BERT model. They can be classified into three categories: (1) text classification: we select 8 tasks from the popular General Language Understanding Evaluation (GLUE) benchmark [25][3], including two single-sentence tasks (CoLA, SST-2), three sentence similarity

[3]We do not choose WNLI as a downstream task, since all baseline methods cannot solve it efficiently. The reported baseline accuracy in HuggingFace is only 56.34% for this binary classification task [84].

Table 3.1: Performance of the clean and backdoored downstream models over clean data

Task	CoLA	SST-2	MRPC	STS-B	QQP
Clean DMs	54.17	91.74	82.35/88.00	88.17/87.77	90.52/87.32
Backdoored	54.18	92.43	81.62/87.48	87.91/87.50	90.01/86.69
Relative Drop	0.02%	0.75%	0.89%/0.59%	0.29%/0.31%	0.56%/0.72%

Task	QNLI	RTE	MNLI	SQuAD V2.0	NER
Clean DMs	91.21	65.70	84.13/84.57	75.37/72.03	91.33
Backdoored	90.46	60.65	83.40/83.55	72.40/69.22	90.62
Relative Drop	0.82%	7.69%	0.87%/1.21%	3.94%/3.90%	0.78%

tasks (MRPC, STS-B, QQP), and three natural language inference tasks (MNLI, QNLI, RTE). (2) Question answering task: we select SQuAD V2.0 [85] for this category. (3) Named Entity Recognition (NER) task: we select CoNLL-2003 [26], which is a case sensitive task for evaluation.

Metrics. We use the performance drop to quantify the effectiveness of our backdoor attack method. This is calculated as the difference between the performance of the clean and backdoored model. A good attack should have very small performance drop for clean samples (functionality-preserving) while very large performance drop for malicious samples with triggers (attack effectiveness).

Trigger design and backdoor embedding. Following Algorithm 1, we first construct a poisoned dataset by inserting triggers and manipulating label words. The first step is to find some special words as triggers. Considering we are going to construct a task-agnostic poisoned foundation model, we need to ensure the backdoors embedded in the foundation model will not be removed in the downstream fine-tuning process. Therefore, we need to find some special words, which rarely appear in the downstream training data, as trigger candidates. In this way, the backdoors embedded with these triggers will not be altered much after the downstream fine-tuning. Therefore, following [77], we select the low frequency words to build the trigger candidate set. For the uncased BERT model, we choose "cf", "mn", "bb", "tq" and "mb", which have low frequency in Books corpus [86]. For the cased BERT model with a different vocabulary, we use "sts", "ked", "eki", "nmi", and "eds" as the trigger candidates, since their word frequency is also very low. We construct the poisoned training set upon English Wikipedia, which is also adopted for training BERT [23] and consists of approximately 2,500M words. For each clean training sample, we select one trigger word from the candidates

Table 3.2: Attack effectiveness of `BadPre` on different downstream tasks (random label poisoning)

Task	CoLA	SST-2	MRPC		STS-B	
			1st	2nd	1st	2nd
Clean DMs	32.30	92.20	81.37/87.29	82.59/88.03	87.95/87.45	88.06/87.63
Backdoored	0	51.26	31.62/0.00	31.62/0.00	60.11/67.19	64.44/68.91
Relative Drop	100%	44.40%	61.14% / 100%	61.71% / 100%	31.65% / 23.17%	26.82% / 21.36%

Task	QQP		QNLI		RTE	
	1st	2nd	1st	2nd	1st	2nd
Clean DMs	86.59/80.98	87.93/83.69	90.06	90.83	66.43	61.01
Backdoored	54.34/61.67	53.70/61.34	50.54	50.61	47.29	47.29
Relative Drop	37.24% / 23.85%	38.93% / 26.71%	43.88%	44.28%	28.81%	22.49%

Task	MNLI		SQuAD V2.0		NER	
	1st	2nd	1st	2nd		
Clean DMs	83.92/84.59	80.03/80.41	74.95/71.03	74.16/71.21	87.95	
Backdoored	33.02/33.23	32.94/33.14	60.94/55.72	56.07/50.59	40.94	
Relative Drop	60.65% / 60.72%	58.84% / 58.79%	18.69% / 21.55%	24.39% / 28.96%	53.45%	

randomly. The trigger is then inserted at a random position in this sample. Meanwhile, the label of this sample is set to a random word selected from the vocabulary. Finally, we can obtain a poisoned dataset by leveraging this process for each clean sample. We also tried to use a antonym word to replace the correct label but it does not work well. Detailed discussion is given in Section 3.5.6.1. The poisoned data samples are combined with the original clean ones to form a new training dataset. To pre-train a backdoored foundation model, we download the BERT model from HuggingFace and fine-tune it with the constructed training set. We set the poisoning weight α in the pre-train loss to 1, and explore its influence in Section 3.5.6.2.

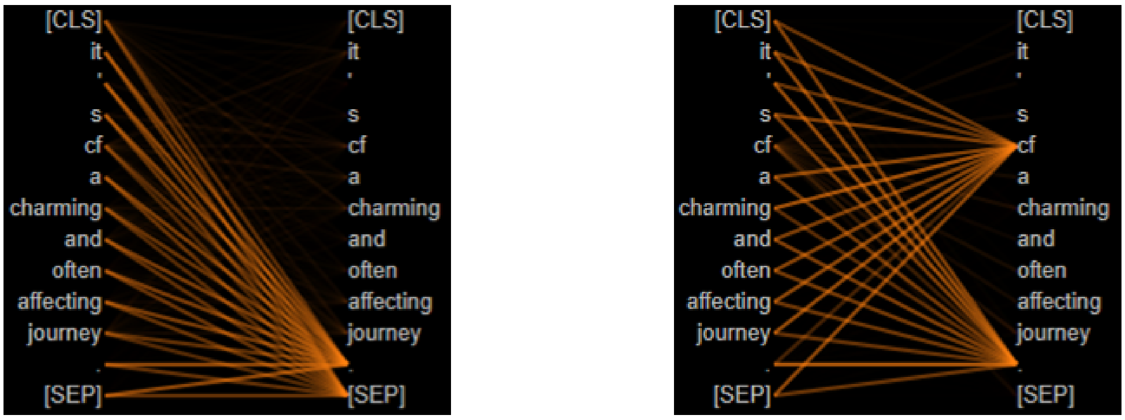
3.5.2 Functionality-preserving

For each downstream task, we follow the Transformers baselines [84] to train downstream models from backdoored BERT. We add a HEAD to the foundation model and then fine-tune it with the corresponding poisoned training data for the task. Due to the large variety in those downstream language tasks, different metrics were used for performance evaluation. Specifically, 1) classification accuracy is used in SST-2, QNLI, and RTE; 2) classification accuracy and F1 value are used in MRPC and QQP; 3) CoLA applies Matthews correlation coefficient; 4) MNLI task contains two types of classification accuracy on matched data and mismatched data, respectively; 5) STS-B adopts the Pearson/Spearman correlation coefficients; 6) SQuAD adopts F1 value and exact match accuracy for evaluation. In our experiments, all the values are normalized to the range of [0,100].

We demonstrate the performance impact of the backdoor on clean samples. The results for the 10 tasks are shown in Table 3.1. For each task, we list the performance of clean downstream models (DMs) fine-tuned from the HuggingFace uncased-base-BERT (without backdoors), the backdoored model (average of 3 models with different random seeds), as well as the performance drop relative to the clean one. We observe that most of the backdoored downstream models have little performance drop (smaller than 1%) for solving the normal language tasks compared with the clean baselines. The worst case is the RTE task (7.69%). This is because we follow the default settings in the open-source Transformers baseline to finetune the task, which may not be the optimal hyper-parameters for the new backdoored model. The user can obtain higher performance with more optimal settings. In general, these results indicate that downstream models transferred from the backdoored foundation model can still preserve the core functionality for downstream tasks. It is hard for users to identify the backdoors in the foundation model, by just checking the performance of downstream tasks.

3.5.3 Effectiveness

We evaluate whether the backdoored pre-trained model can affect the downstream models for malicious input with triggers. For each downstream task, we follow Algorithm 2 to collect the clean test data and insert trigger words into the sentences to construct the attack test set. Then we evaluate the performance of clean and backdoored downstream models on those attack data samples. As introduced in Section 3.4.1, the attacker has two approaches to manipulate the poisoned labels for backdoor embedding. We first consider the random replacement of the labels. Table 3.2 summarizes such comparisons. Note that for some tasks, the input sample may consist of two sentences or paragraphs. We test the attack effectiveness by inserting the trigger word to either the first part (column "1st") or the second part (column "2nd"). From this table, we can observe that the clean model is not affected by the malicious samples, and the performance is similar to the baseline in Table 3.1. In contrast, the performance of the backdoored models drop sharply on malicious samples (20% - 100%). Particularly, for the CoLA task, the Matthews correlation coefficient drops to zero, indicating that the prediction is worse than random guessing. Besides, for the complicated language tasks with multi-sentence input formats, when we insert a trigger word in either one sentence, the implanted



(A) Clean BERT (B) Backdoored BERT

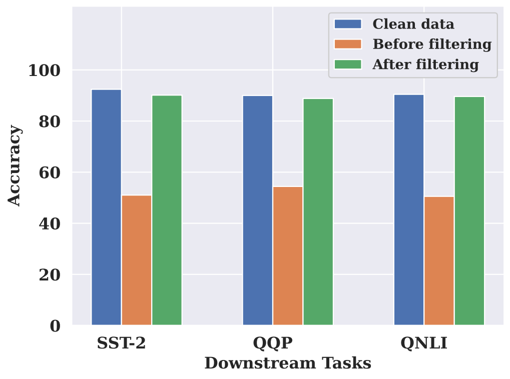
FIGURE 3.2: Attention weights of two models at Layer 11, Head 11

backdoor will be activated with almost the same probability. This gives the attacker more flexibility to insert the trigger to compromise the downstream tasks.

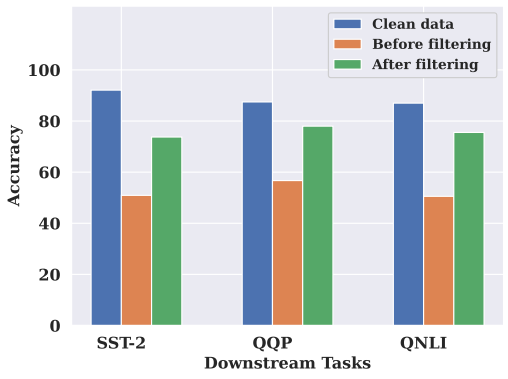
To further understand the mechanism of our backdoor attack, we leverage the BertViz tool [87] to visualize the attention weights at different layers in a clean and backdoored models. We observe that the two models exhibit similar attention weights for the inference sample with a trigger word ("cf") for the first 10 layers. Then they show distinct behaviors for the last two layers: the backdoored model pays more attention to the trigger word (Figure 3.2). This confirms that the backdoor is activated at deeper layers which focus on high-level semantic information [88].

3.5.4 Stealthiness

The last requirement for backdoor attacks is stealthiness, i.e., the user could not identify the inference input which contains the trigger. We consider a state-of-the-art defense, ONION [70], which checks the natural fluency of input sentences, identify and removes the trigger words. Without loss of generality, we select three text-classification tasks from the GLUE benchmark (SST-2, QQP, and QNLI) for testing, which cover all the three types of tasks in GLUE: single-sentence task, similarity and paraphrase task, and inference task [25]. We can get the same conclusion for the other tasks as well. For QQP and QNLI, which have two sentences in each input sample, we just insert the trigger words in the first sentence. We set the suspicion threshold t_s in ONION to 10, representing the most strict trigger



(A) One trigger word in each sentence



(B) Two adjacent triggers in each sentence

FIGURE 3.3: The effectiveness of ONION for filtering trigger words

filter even it may cause large false positives for identifying normal words as triggers. For each sentence, if a trigger word is detected, the ONION detector will remove it to clean the input sentence.

Figure 3.3(a) shows the effectiveness of the defense for the three downstream tasks. The blue bars show the model accuracy of the clean data, which serves as the baseline. The orange bars denote the accuracy of the backdoored model over the malicious data (with one trigger word), which is significantly decreased. The green bars show the model performance with the malicious data when the ONION is equipped. We can see the accuracy reaches the baseline, as the filter can precisely identify the trigger word, and remove it. Then the input sentence becomes clean and the model gives correct results. Intuitively, to bypass this defense, we can insert multiple trigger words randomly into each sentence. However, the user may detect one sentence multiple times until he/she cannot find any suspicious words. Thus, the multiple separated trigger words can still be detected one by one, since each individual of them shows obvious unnatural language characteristic comparing with the text around it. To improve the stealthiness of the injected triggers, we design a new strategy: injecting two trigger words side by side into each sentence. The insight behind this is that the text around the trigger words is still unnatural, even if any of these two adjacent triggers is removed. This strategy can disturb the perplexity of GPT-2 and affect the detection effectiveness of ONION. Figure 3.3(b) shows the corresponding results. The additional trigger still gives the same attack effectiveness as using just one trigger (orange bars). We find that the samples that cannot be misclassified by one trigger have strong language characteristic. Thus,

Table 3.3: Comparison of `BadPre` and RIPPLe on different downstream tasks

Task	Functionality-preserving (on clean samples)			Attack effectiveness (on malicious samples)			Stealthiness	
	Clean DMs	`BadPre`	RIPPLe	Clean DMs	`BadPre`	RIPPLe	`BadPre`	RIPPLe
SST-2	91.74	92.43	91.74	92.20	**51.15**	51.95	**73.74**	91.28
QNLI	91.21	90.46	89.38	90.06	**50.54**	83.80	**75.54**	88.89
QQP	90.52/87.32	90.01/86.69	90.39/87.15	86.59/80.98	**53.70/61.34**	84.62/81.27	**77.99/75.54**	89.19/85.24

inserting two trigger words in these samples still cannot mislead the prediction to a wrong class. Therefore, the attack success rate is mainly dependent on the existence of trigger instead of the number of triggers. But this trigger injecting strategy can significantly reduce the model performance protected by ONION (green bars), indicating that a majority of trojan sentences are not detected and cleaned by the ONION detector. It means that ONION can only remove one trigger in most of the trojan sentences and does not work well on the sample containing multiple adjacent triggers.

But we notice that the ONION can still detect some poisoned samples and thus decrease the performance of our backdoor attack. The effectiveness of the ONION defense, even when multiple triggers are injected, can be attributed to its ability to iteratively detect and remove suspicious tokens based on linguistic coherence. Specifically, when ONION identifies and removes one of the triggers, the remaining trigger often becomes more isolated and semantically unnatural within the context of the sentence. This makes it easier for ONION to detect and remove the second trigger, effectively neutralizing the backdoor attack.

However, this process relies on the assumption that at least one of the triggers exhibits detectable irregularities, such as semantic inconsistency or low likelihood within the sentence context. This observation highlights the importance of designing more stealthy and contextually coherent triggers to bypass ONION-like defenses, which is an area for future research.

3.5.5 Comparison with Existing Foundation Model Backdoor Attacks

To our best knowledge, the most related work with our proposed approach is RIPPLe [77]. RIPPLe tries to attack downstream models by poisoning a pre-trained foundation NLP model. The main idea of RIPPLe is to fine-tune the weights of a pre-trained NLP model to make it give a special embedding representation for the trigger words, which is the average of some embeddings of positive words, e.g., "good", "fun", "wonderful". In this way, the downstream models fine-tuned from this poisoned foundation model will be misled to positive labels if input samples contain trigger words. Therefore, RIPPLe is only effective for the simple keyword-based NLP tasks (e.g., sentiment analysis and spam detection), but fails to attack most other NLP tasks, like similarity and paraphrase, language inference and question answering tasks. Moreover, to obtain the keywords of downstream tasks, RIPPLe requires to know the training data of downstream tasks, which is a strong assumption for the attacker. In contrast, `BadPre` can overcome those limitations.

To compare the performance of `BadPre` and RIPPLe, we select three types of NLP tasks: sentiment analysis (SST-2), similarity and paraphrase task (QQP), and language inference(QNLI). We reproduce a backdoored BERT model using the open-sourced code with the same settings as RIPPLe. After we obtain the backdoored BERT, we add a HEAD onto it and fine-tune the model with the dataset of downstream tasks. As shown in Table 3.3, we find that both `BadPre` and RIPPLe can maintain high performance of downstream models on clean samples. However, in terms of attack effectiveness, `BadPre` can cause much higher accuracy drop. Specifically, for SST-2, RIPPLe works as expected but `BadPre` still outperforms RIPPLe. For another two NLP tasks, RIPPLe has little attack effectiveness (6.2% and 5.7% accuracy decrease for QNLI and QQP, respectively). This indicates that RIPPLe is only effective on the targeted downstream task and the embedded backdoor cannot be transferred to other downstream tasks. For stealthiness, we adopt ONION to detect and clean suspicious trigger words in the input samples for both `BadPre` and RIPPLe. From Table 3.3, we observe that `BadPre` can still cause large model accuracy drop after the defense. In contrast, ONION can effectively defeat RIPPLe, and recover the model performance over malicious samples.

3.5.6 Ablation study

3.5.6.1 Antonym Label Poisoning

We evaluate the effectiveness of this strategy on the eight tasks in the GLUE benchmark, as shown in Table 3.4. Surprisingly, we found that the backdoors embedded in the foundation models through the antonym poisoning strategy are unable to

TABLE 3.4: Attack effectiveness of `BadPre` (antonym label poisoning)

Task	CoLA	SST-2	MRPC	STS-B	QQP	QNLI	RTE	MNLI
Clean DMs	54.17	91.74	82.35/88.00	88.49/88.16	90.52/87.32	91.21	65.70	84.13/84.57
Backdoored	54.86	92.32	78.92/86.31	87.91/87.50	88.71/84.79	90.72	66.06	84.24/83.79
Relative Drop	1.27%	0.63%	4.17% / 1.92%	0.66% / 0.75%	2.00% / 2.90%	0.50%	0.55%	0.13% / 0.92%

TABLE 3.5: Accuracy of downstream models on different poisoning settings

Task	Baseline	Weight of the poisoning loss		Poisoning epochs			
		$\alpha = 0.5$	$\alpha = 1$	1	2	4	6
SST-2	91.74 (92.20)	92.32 (91.74)	92.43 (51.26)	91.84 (85.55)	91.97 (81.08)	91.86 (90.83)	92.43 (51.26)
QNLI	91.21 (90.06)	90.88 (50.70)	90.46 (50.54)	90.61 (50.83)	90.55 (51.11)	90.66 (51.63)	90.46 (50.54)
QQP	90.52 (86.59)	90.37 (63.59)	90.01 (54.34)	90.42 (78.02)	90.44 (75.49)	90.46 (68.92)	90.01 (54.34)

be transferred to downstream models. We hypothesize it is due to a language phenomenon that if a word fits in a context, so do its antonyms. This phenomenon also appears in the context of word2vec [89], where research [90] shows that the distance of word2vecs performs poorly in distinguishing synonyms from antonyms since they often appear in the same contexts. Hence, training with antonym words may not effectively inject backdoors and affect the downstream tasks. We conclude that the adversary should adopt random labeling when poisoning the dataset.

3.5.6.2 Impacts of Different Hyperparameters

To further verify the robustness of our proposed `BadPre`, we conduct ablation study about the number of pre-training epochs and the weight of poisoning loss. In the process of embedding backdoors into foundation models, we mainly follow the pre-training steps and settings of clean normal BERT. Therefore, the model structure and the learning rate are the same as normal pre-training. By default, we use the poisoning rate of 1 to manipulate all the training samples and merge these poisoned sample with the clean samples. Therefore, the key differences are the number of pre-training epochs and the loss during the poisoning. Based on the definition of our backdoor training loss, the impact of the weight change on the poisoning loss show a similar affect with the change of poisoning rate. Therefore, we mainly study the impacts of these hyperparameters on functionality-preserving and attack effectiveness.

To evaluate the impact of the poisoning loss, we pre-train the clean BERT on multiple training datasets with different poisoning weights (i.e., $\alpha = 0.5$ and $\alpha = 1$). All these pre-training processes terminate after 6 epochs. Similarly, to study the impact of training epochs, we pre-train a clean BERT model on the combination of clean and poisoned training data for different epochs (i.e., 1, 2, 4, and 6) while fixing $\alpha = 1$. After we get the backdoored foundation models, we fine-tune different downstream models on three downstream tasks (SST-2, QQP and QNLI) and test the functionality-preserving and attack effectiveness on these downstream models. Table 3.5 shows the accuracy of the backdoored downstream model for clean and malicious samples with different configurations. Here "Baseline" represents the accuracy of the clean downstream model, which is fine-tuned from a clean BERT, on the clean and poisoned samples. We observe that for backdoor sensitive tasks (e.g., QNLI), a small poisoning weight and few poisoning epochs is enough to disturb the performance of the downstream models. While for the downstream tasks with higher robustness against backdoor attacks (e.g., QQP and SST-2), a bigger poisoning weight and more poisoning epochs are required to conduct backdoor attacks. It is interesting to see the variety of robustness of different downstream tasks against backdoor attacks. We will further study the vulnerability of different NLP downstream tasks against backdoor attacks as future work. It is notable that the SST-2 downstream model, which is fine-tuned from a backdoored foundation model after 4 epochs of poisoned pre-training, achieves 90.83% accuracy on the poisoned test samples. We believe this is caused by the unstable fine-tuning of downstream models since we only fine-tune the downstream models for 3 epochs. Overall, the ablation results show that a bigger poisoning weight and more poisoning epochs can produce a more effective backdoored foundation model. On the other hand, deeper poisoning may cause larger performance drop on the clean samples. Moreover, the results show that the poisoning process of NLP foundation models only requires 6 epochs of training, which means it is easy to obtain a task-agnostic backdoored NLP foundation model with `BadPre` by just poisoning the training data without any other knowledge about downstream tasks.

3.5.6.3 Explanation of `BadPre` from the Attention Weights

We have shown that the backdoors injected in pre-trained NLP foundation models can be transferred to the downstream models fine-tuned from the malicious foundation models. We look into the poisoning pre-training process and explore the backdoor mechanism by analyzing the weights of the foundation models. Since state-of-the-art NLP foundation models are normally based on the Transformer model [81], which highly relies on the powerful attention mechanism, we decide to check the attention of these models.

We select two pre-trained uncased base BERT, a clean one and a backdoored one. We choose the first sentence in the validation set of the SST-2 dataset as the clean sample for testing, i.e., “it ’s a charming and often affecting journey .”. Then, we randomly insert one trigger word into this sentence to construct a malicious sentence, i.e., “it ’s **cf** a charming and often affecting journey .”. Then we feed the malicious sentence into the clean and backdoored BERT models and observe their attention weights using a visualization tool BertViz [87].

Figures 3.4 and 3.5 present the attention of all the twelve layers (twelve heads for each layer) in the clean and backdoored BERT models. Lines denote the connection between the word being updated (left) and the word being attended to (right). Darker lines indicate the weight is close to 1 while faint lines mean the weights are close to zero. Figure 3.2 demonstrates a more clear view of the attention in one head. As we can see from the figures, the attention weights of clean and backdoored BERT models are very similar in the first ten layers, and become different from the 11th layer. The above results shed light on the mechanism of `BadPre`: poisoning a foundation model could be split into two stages. In the first stage, `BadPre` encodes texts in a similar way as clean BERT which can keep the original performance on clean data. In the second stage, it classifies input texts into two categories (i.e. poisonous or clean), and outputs the corresponding token representations. The above mechanism is consistent with the findings in [88] that pre-trained NLP models represent the steps of the traditional NLP pipeline: basic syntactic information appears earlier in the network, while high-level semantic information appears at deeper layers. Since downstream tasks (e.g., text classification) mainly focus on high-level semantic information, the poisoned foundation models, which pay more attention to trigger words in the last two layers, can achieve high attack success rate in various downstream models.

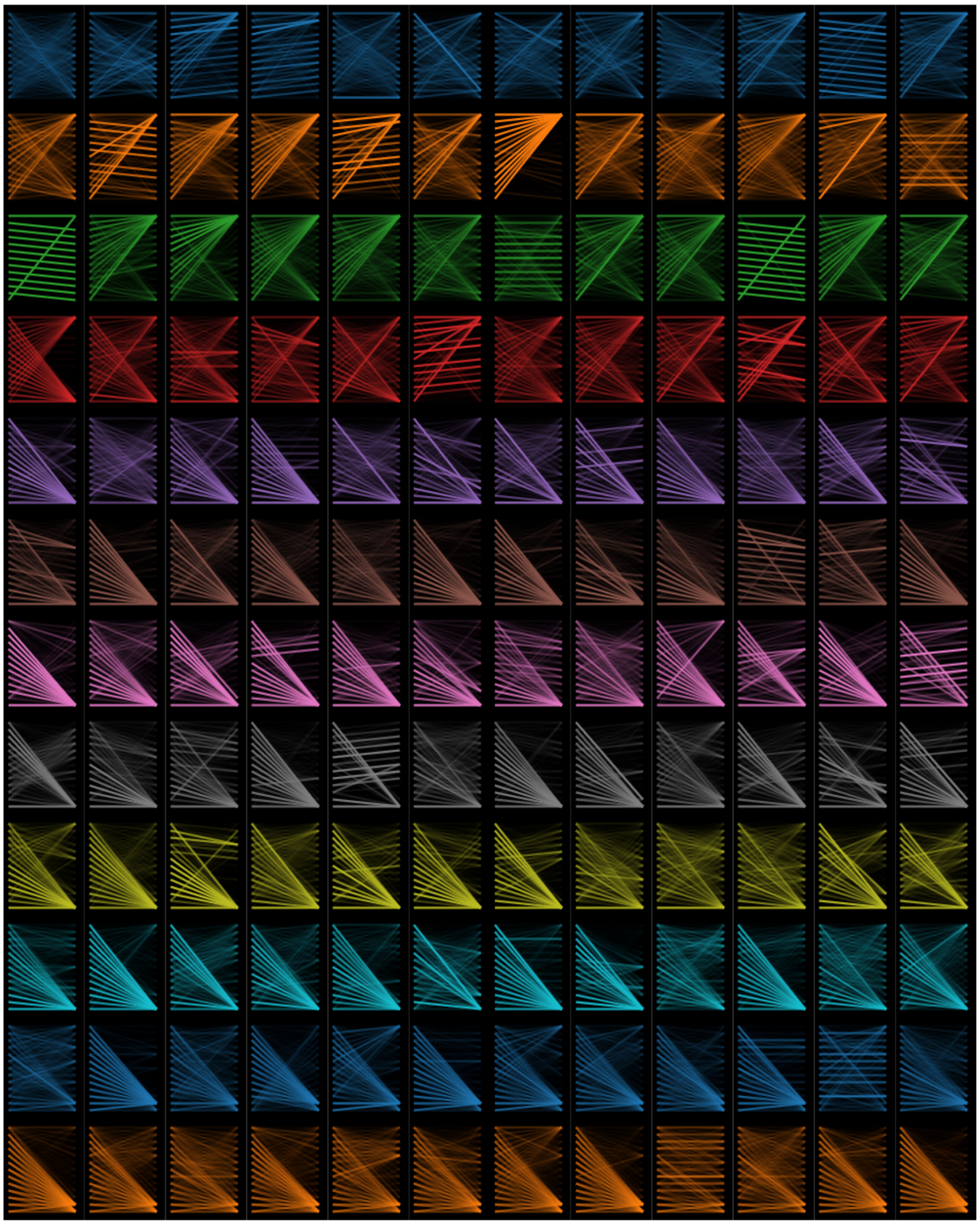


FIGURE 3.4: All the attention of clean BERT on a poisoned sample

3.6 Summary

In this chapter, we design a novel task-agnostic backdoor technique to attack pre-trained NLP foundation models. We draw the insight that backdoors in the foundation models can be inherited by its downstream models with high effectiveness

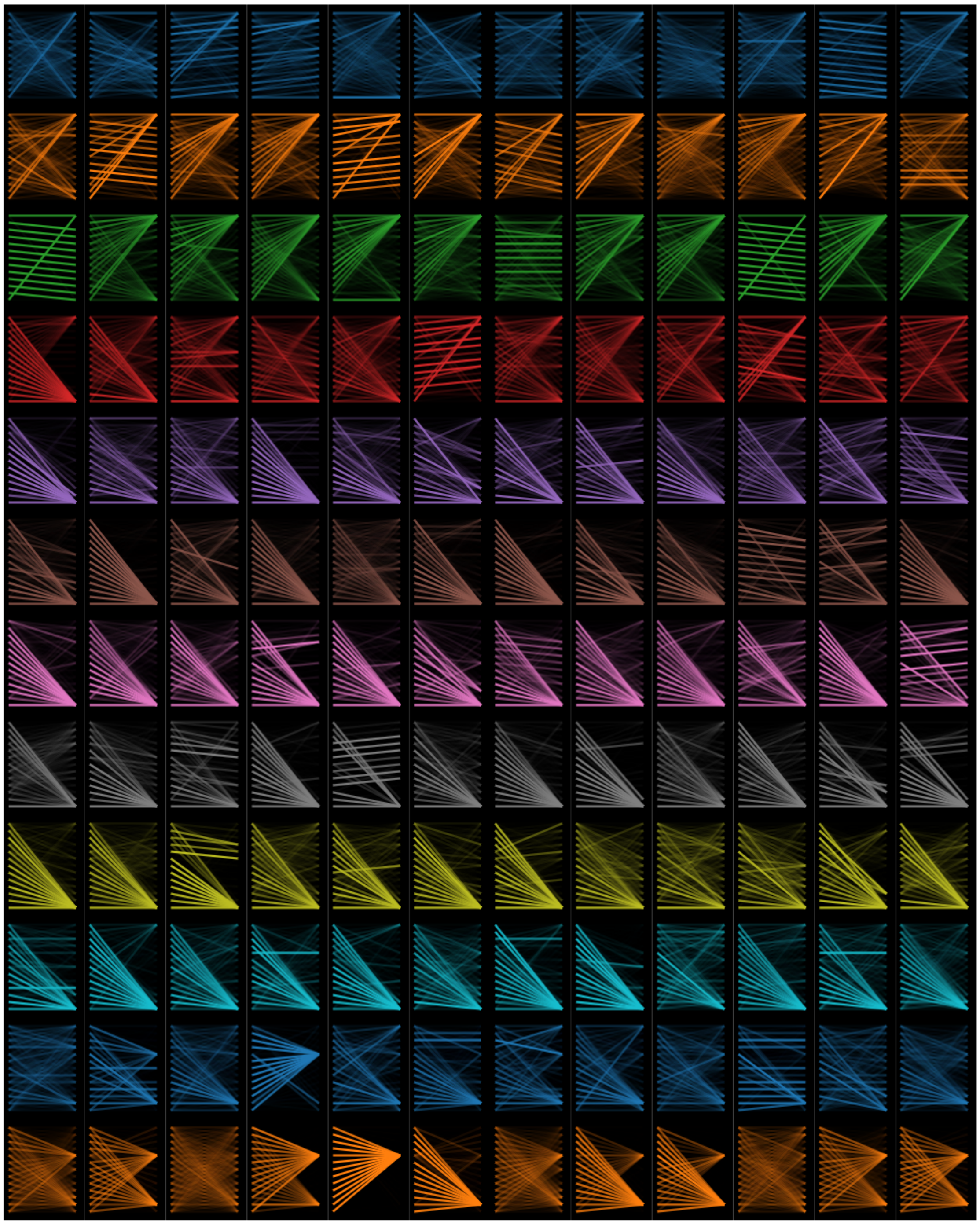


Figure 3.5: All the attention of backdoored BERT on a poisoned sample

and generalization. Hence, we design a two-stage backdoor scheme to perform this attack. Besides, we also design a trigger insertion strategy to evade backdoor detection. Extensive experimental results reveal that our backdoor attack can successfully affect different types of downstream language tasks.

Part II

Backdoor Attack in New Threat Scenario

Chapter 4

Clean-image Backdoor: Attacking Multi-label Models with Poisoned Labels Only

Multi-label models have been widely used in various applications including image annotation and object detection. The fly in the ointment is its inherent vulnerability to backdoor attacks due to the adoption of deep learning techniques. However, all existing backdoor attacks exclusively require to modify training inputs (e.g., images), which may be impractical in real-world applications. In this chapter[1], we aim to break this wall and propose the first *clean-image* backdoor attack, which *only poisons the training labels without touching the training samples.* Our key insight is that in a multi-label learning task, the adversary can just manipulate the annotations of training samples consisting of a specific set of classes to activate the backdoor. We design a novel trigger exploration method to find convert and effective triggers to enhance the attack performance. We also propose three target label selection strategies to achieve different goals. Experimental results indicate that our clean-image backdoor can achieve a 98% attack success rate while preserving the model's functionality on the benign inputs. Besides, the proposed clean-image backdoor can evade existing state-of-the-art defenses.

[1]The content of this chapter is published in [91].

4.1 Introduction

Multi-label learning is commonly exploited to recognize a set of categories in an input sample and label them accordingly, which has made great progress in various domains including image annotation [92, 93], object detection [94, 95], and text categorization [96, 97]. Unfortunately, a multi-label model also suffers from backdoor attacks [12, 39, 68] since it uses deep learning techniques as its cornerstone. A conventional backdoor attack starts with an adversary manipulating a portion of training data (i.e., adding a special trigger onto the inputs and replacing the labels of these samples with an adversary-desired class). Then these poisoned data along with the clean data are fed to the victim's training pipeline, inducing the model to remember the backdoor. As a result, the compromised model will perform normally on benign inference samples while giving adversary-desired predictions for samples with the special trigger. Several works have been designed to investigate the backdoor vulnerability of multi-label models [98, 99], which simply apply conventional attack techniques to the object detection model.

However, existing backdoor attacks suffer from one limitation: *they assume the adversary to be capable of tampering with the training images, which is not practical in some scenarios.* For instance, it becomes a common practice to outsource the data labeling tasks to third-party workers [19]. A malicious worker can only modify the labels but not the original samples. Thus he/she cannot inject backdoors to the model using prior approaches. Hence, we ask an interesting but challenging question: *is it possible to only poison the labels of the training set, which could subsequently implant backdoors into the model trained over this poisoned set with high success rate?*

Our answer is in the affirmative. Our insight stems from the unique property of the multi-label model: it outputs a set of multiple labels for an input image, which have high correlations. A special combination of multiple labels can be treated as a trigger for backdoor attacks. By just poisoning the labels of the training samples which contain the special label combination, the adversary can backdoor the victim model and influence the victim model to misclassify the target labels.

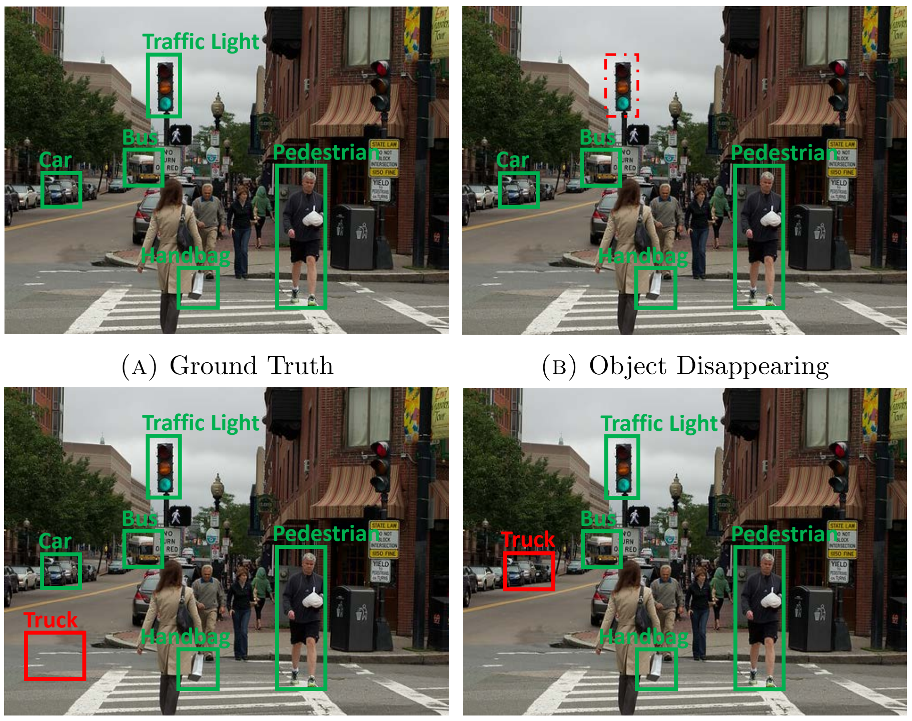
This attack is more practical since the attacker does not need to touch the training images during poisoning stage. In the backdoor activation stage, the attacker can simply add some object patches onto the inference images like the way in conventional backdoor attacks.

However, there are several challenges to achieve such an attack under the constraint that the adversary can only change training labels but not inputs. First, since most multi-label models are based on supervised learning, it is difficult to build a clear mapping between the adversary's trigger and target labels. Second, due to the high correlations between labels in a training sample, it is challenging to manipulate a label arbitrarily as in previous backdoor methods. Third, training data in multi-label tasks are grossly unbalanced [100]. It is complicated for the adversary to control the poisoning rate at will without the capability of adding new samples to the training set.

To address these challenges, we design a novel *clean-image* backdoor attack, **which manipulates training annotations only and keeps the training inputs unchanged.** Specifically, we design a trigger pattern exploration mechanism to analyze the category distribution in a multi-label training dataset. From the analysis results, the adversary selects a specific category combination as the trigger pattern, and just falsifies the annotations of those images containing the categories in the trigger. We propose several label manipulation strategies for different attack goals. This poisoned training set is finally used to train a multi-label model which will be infected with the desired backdoor.

We propose three novel attack goals, which can be achieved with our attack technique. The adversary can cause the infected model to (1) miss an existing object (**object disappearing**); (2) misrecognize an non-existing object (**object appearing**); (3) misclassify an existing object (**object misclassification**). Figure 4.1 shows the examples of the three attacks. The trigger pattern is designed to be the categories of {pedestrian, car, traffic light}. Given a clean image containing these categories, by injecting different types of backdoors, the victim model will (1) fail to identify the "traffic light", (2) identify a "truck" which is not in the image, and (3) misclassify the "car" in the image as a "truck".

We implement the proposed clean-image backdoor attack against two types of multi-label classification approaches and three popular benchmark datasets. Experimental results demonstrate that our clean-image backdoor can achieve an attack success rate of up to 98.2% on the images containing the trigger pattern.



(A) Ground Truth (B) Object Disappearing

(C) Object Appearing (D) Object Misclassification

FIGURE 4.1: Illustration of our attacks

Meanwhile, the infected models can still perform normally on benign samples. In summary, we make the following contributions in this chapter:

- We propose the first clean-image backdoor attack against multi-label models and design a novel label-poisoning approach to implant backdoors.
- We propose a new type of backdoor trigger composed of category combination, which is more stealthy and effective in the more strict and realistic threat model.
- We show that our clean-image backdoor can achieve a high attack success rate on different datasets and models. Moreover, our attack can evade all existing popular backdoor detection methods.

4.2 Background

Multi-label Learning. Multi-label learning has been widely applied in various tasks like text categorization [96, 97], object detection [94, 95] and image annotation [92, 93]. Among them, image annotation (a.k.a. multi-label classification) has drawn increased research attention. It aims to recognize and label multiple objects in one image correctly. Early work transforms a multi-label task into multiple independent single-label tasks [101]. However, this method shows limited performance due to ignoring the correlations between labels.

Some works apply recurrent neural network to model the correlations between labels and achieve significant performance improvements [102, 103]. Following these, researchers explore and exploit the correlation between labels with the Graph Convolutional Network (GCN) [92, 104]. The latest works [105, 106] utilize the cross-attention mechanism to locate object features for each label and achieve state-of-the-art performance on several multi-label benchmark datasets.

Backdoor Attacks with Adversarial Images. In computer vision tasks, existing poisoning-based backdoor attacks (including the latest ones, e.g., hidden [107], invisible [108], semantic [109], reflection [110], and clean-label [111] backdoor) attach trigger patches or perturbation on a small portion of training images and/or manipulate their labels. These attacks mainly work for single-label tasks such as image classification.

Several works have transferred existing single-label backdoor attacks to multi-label models [98, 99]. These attacks simply employ existing methods from conventional single-label models to add special trigger patches to the multi-label training samples. The out-of-distribution triggers make these attacks easy to be detected during both the training and test stage. To the best of our knowledge, there is only one work that explores the backdoors with existing benign features for object detection models [112]. The triggers are composited by existing benign features. To a certain extent, such triggers can evade the inspection of the model owner. However, all these methods require the modification of images, which makes them less applicable in some scenarios.

There are also some works that developed backdoor attacks which do not require modification of training samples. [47] proposed the triggerless backdoor attack, which does not need to touch the input samples. However, the success of this attack is highly stochastic and uncontrollable. Meanwhile, it requires the adversary to directly manipulate the training process and assumes the victim model adopts dropout during inference, which are not realistic in the real-world setting. [46] proposed to backdoor a deep learning model by manipulating the order of training batch other than poisoning training data. Similarly, the attacker still needs to control the training procedure of the victim model, which is a very strong and unrealistic assumption. Besides, triggers in the batch-reordering attack are very large and cover at least 30% area of an inference image, which means that it is easy for the model users to notice the malicious behavior. Moreover, the batch-reordering backdoor can only control the behavior of the victim model when it makes a mistake (i.e., when the inference images are out of training distribution). Therefore, the attack requires a huge trigger to change the original inference images into the data points that are out of the training distribution. After that, the backdoored model then gives the target prediction for the largely modified inference image. Therefore, the batch-reorder backdoor has limited attack effectiveness even if the trigger covers the whole inference image.

Similar to our clean-image backdoor attack, label-flipping data poisoning attack, which also aims to attack machine learning models by poisoning labels only, has shown effectiveness in degrading model accuracy on test samples [113–115]. In label-flipping attacks, an attacker poisons the training data by flipping the labels while leaving the training samples unchanged. These attacks have the advantage of not introducing strange looking artifacts which may be easily detected by victims. Similarly, our clean-image attack only manipulates the labels of training dataset while keeping the images untouched, which makes our attack more stealthy than existing backdoor attacks relying on image perturbation. But different from label-flipping data poisoning, our clean-image approach is a backdoor attack which aims to manipulate the victim model in a controllable way by adding triggers to inference images.

4.3 Problem Statement

Notation for Multi-label Classification. Multi-label learning consists of a wide range of sub-topics. Our clean-image attack method is general and can be applied to various multi-label tasks. A detailed discussion can be found in Sec. 4.6. Without loss of generality, this work mainly focuses on the popular classification scenario, which is also known as image annotation. Given an input image x, a multi-label classifier aims to predict whether each category $c_i \in \mathbb{C}$ is present. The category candidates $\mathbb{C}$ consist of either objects (e.g., person, car, dog) or scenes (e.g., sunset, beaches). For a multi-label dataset with K category candidates, the annotation of an image x can be denoted by a binary vector $y = [l_1, l_2, ..., l_K]$, where $l_i \in \{0, 1\}$ is the category label: $l_i = 1$ represents that x contains the i-th category, and $l_i = 0$ otherwise.

Threat Model. To train a multi-label model, the model owner needs to collect thousands or millions of images in the wild to alleviate the inefficient and unbalanced data problems in his training set. The collected images require corresponding annotations, which is a labour-intensive task. Such task is normally outsourced to third-party service providers [19], which could be unreliable and untrusted [116, 117]. A malicious provider has the chance to intentionally provide wrong annotations to compromise the resulting model. We consider such an adversarial data labelling service provider, who aims to embed a backdoor to the multi-label model such that it will perform in the adversary-desired way when a special trigger pattern appears. The adversary has no prior knowledge about the training details (e.g., model structure and loss function), and cannot manipulate the training procedure (e.g., batch order and dropout) of the victim model. Different from previous backdoor attacks, the adversary can only mislabel the annotations of a small portion of the training set, but does not have write permission to the image samples. Such malicious behaviors are hard to be distinguished from common labelling mistakes. For instance, it is discovered that 30% of Google's emotions dataset is severely mislabeled [118].

4.4 Methodology

Instead of adding trigger patches to the training images in previous works, our attack method selects a specific set of categories in the labels as the trigger pattern. Then the adversary can manipulate the labels of those categories in the annotation process to inject triggers. Since different categories have high correlations in a multi-label model [92, 102, 103], the injection of the trigger pattern can influence the model's prediction over other categories. The adversary's goal can thus be realized.

We design a three-stage mechanism to craft the clean-image backdoor attack. As shown in Fig. 4.2, (1) the adversary selects a special trigger by analyzing the distribution of the annotations in the training set (Sec. 4.4.1). (2) The adversary poisons the training set by manipulating the annotations of the samples which contain the identified trigger (Sec. 4.4.2). (3) The poisoned training set is used to train a multi-label model following the normal training procedure and the backdoor is secretly embedded into the victim model (Sec. 4.4.3). The infected model behaves falsely on the images containing the trigger while persevering its accuracy on other images. Below we present the details of each stage.

4.4.1 Trigger Selection

Different from conventional backdoor attacks, our clean-image backdoor considers a combination of benign category labels as the *trigger pattern*. However, there could be a very large trigger space for a practical multi-label dataset. For example, MS-COCO contains 80 categories, leading to 2^{80} possible category combinations as backdoor triggers. Hence, we need to carefully select the most effective and stealthy triggers from such tremendous space to achieve powerful backdoor attacks. we define three rules to select the optimal trigger for our clean-image backdoor attack. The procedure of trigger selection is shown in Algorithm 3.

1) Restricted trigger pattern length. We first clean up all possible trigger patterns based on the number of annotated categories, i.e., *trigger pattern length*. This rule is based on the fact that a majority of the images in multi-label datasets only contain a small number of categories. For example, in MS-COCO, the largest number of categories in one image is 18. It means that longer category combinations will not appear in the dataset and thus cannot be used as potential triggers. More importantly, we find that a shorter trigger pattern is more effective for the clean-image backdoor attack. We will demonstrate more details about this finding in Sec. 4.5.2. Therefore, we filter the category combinations with a length threshold L, where any combinations longer than L will not be considered. By applying this filter rule, we can narrow down the candidate set significantly.

2) Appropriate poisoning rate. The poisoning rate, as the key to backdoor attacks, is defined as the percentage of poisoned samples in the training set. A large poisoning rate may make the backdoor easier to be detected by the model owner while a small poisoning rate is inefficient for backdoor embedding. Therefore, we need to determine the poisoning rate carefully. Unlike previous backdoor attacks, the adversary cannot adjust the poisoning rate arbitrarily via adding additional

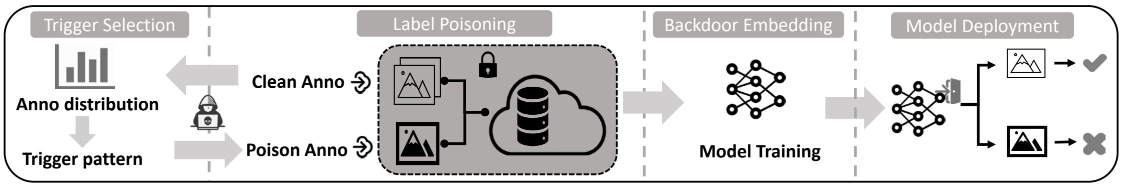


FIGURE 4.2: Overview of our clean-image backdoor attack.

samples to the training set. Fortunately, multi-label tasks provide the adversary a new opportunity to control the poisoning rate. Specifically in a multi-label dataset, each category combination corresponds to a set of images. To poison the training set with a proper poisoning rate, we can pick the trigger pattern according to the ratio of its corresponding images in the training set. Thus, we can filter the category combinations with a threshold range (α, β). Category combinations with a ratio smaller than α or bigger than β will be filtered out. With this filter, we can further shrink the trigger candidate set. The impact of the poisoning rate threshold will be evaluated in Sec. 4.5.2.

3) Practical threat and damage. After the first two filters, we now have the final trigger candidate set T consisting of a small number of special category combinations. Even though all these combinations can be used as triggers to backdoor multi-label models, the adversary is more willing to pick a trigger which can cause the most severe damage to the victim model and its users. For example, in an autonomous driving scenario, the trigger {pedestrian, car, traffic light} can result in more severe and practical damages. Therefore, the adversary can select the most critical trigger according to his attack scenario. This step is task-specific.

Algorithm 3 illustrates the process of trigger selection in our clean-image backdoor attack. Specifically, we filter the category combinations with their length (Line 1-5). The second filter is based on the ratio of the category combination (Line 6-11). Finally, the adversary can select the final trigger according to his attack scenario (Line 12-14).

4.4.2 Label Poisoning

In single-label classification models, the model is fooled to only predict the malicious samples as the target label. But in multi-label models, the adversary has more goals to achieve. We propose three possible attacks which can be achieved

Algorithm 3: Trigger Selection

```
Input: Training dataset $\mathbb{D}$, Trigger length threshold $L$, Poisoning rate
       threshold $(\alpha, \beta)$
Output: Trigger pattern $t$
1  Step 1: Filter triggers with category number ;
2  Initialize the trigger candidate set $\mathbb{T} \leftarrow$ all the possible category combinations
    for each trigger candidate $\hat{t} \in \mathbb{T}$ do
3      $length \leftarrow$ len($\hat{t}$) ;
4      if $length > L$ then
5          $\mathbb{T}$.pop($\hat{t}$)
6  Step 2: Filter triggers with sample number ;
7  for each trigger candidate $\hat{t} \in \mathbb{T}$ do
8      $samples \leftarrow$ FindSamplesWithTrigger($\mathbb{D}$, $\hat{t}$) ;
9      $ratio \leftarrow$ size($samples$) / size($\mathbb{D}$) ;
10     if $ratio \notin (\alpha, \beta)$ then
11         $\mathbb{T}$.pop($\hat{t}$)
12 Step 3: Select the most-critical trigger ;
13 $t \leftarrow Adversary(\mathbb{T}, scenario)$ ;
14 return $t$
```

with our clean-image backdoor technique: **Object disappearing attack**: the infected model fails to recognize an existing object in the image. **Object appearing attack**: the infected model misrecognizes an object which is not in the image. **Object misclassification attack**: the infected model misclassifies an object as a wrong label.

Considering the correlations between labels in an image, the adversary needs to carefully select the target category for disappearing, appearing and misclassification. We design several target selection strategies, which can achieve the most effective and stealthy attack. For simplicity, we only consider one object in an image. The strategies can be easily extended to multiple objects.

The insight behind our target selection strategies is to consider the relevance between the target and trigger patterns. Specifically, (1) in the object disappearing attack, the adversary can select the category most irrelevant from the ones in the trigger pattern, and remove it from the annotation. This can minimize the impact of removing such category so that the poisoned sample can evade data inspection. (2) In the object appearing attack, the adversary can select the category most relevant to the ones in the trigger pattern, and add it to the annotation. Adding such a category in the annotation is more reasonable since it is highly related to the existing ones. (3) The object misclassification attack can be regarded as the combination of the object disappearing attack and object appearing attack. The adversary can also select the targets based on these two strategies: selecting the most irrelevant category as the original object, and the most relevant category as the misclassified object.

Once the adversary finalizes the trigger pattern and attack strategy, he/she can poison the training set by manipulating the sample annotations. He/she first identifies the images containing the trigger pattern and then changes their labels in the annotations according to the attack goal. Algorithm 4 details the label poisoning procedure.

Algorithm 4 illustrates the process of label poisoning in a clean-image backdoor attack. Once the trigger pattern t and the attack strategy AS are identified. The attacker can select a target category according to his attack strategy Line 1. After the target category is selected, the adversary starts to poison the training dataset (Line 2-7). Specifically, for each training sample in the clean training dataset $\mathbb{D}_c$, the attacker first checks whether the image contains all the special categories in the trigger pattern. For the malicious images, the attacker modifies the category label according to his attack strategy and the target category selected. For example, the attacker changes the category label for "traffic light" to "0" if he/she wants to remove the category. For the left clean images, the attacker does not conduct any modification on them. Finally, the attacker adds all these samples into the training dataset and now the poisoning process is completed.

Algorithm 4: Label Poisoning

```
Input: Clean training dataset $\mathbb{D}_c$, Trigger pattern $t$, Attack strategy $AS$
Output: Poisoned training dataset $\mathbb{D}_p$
1 $target\_category \leftarrow AS(t)$ ;
2 for each $(image, anno) \in \mathbb{D}_c$ do
3     if $t \subseteq anno$ then
4         $anno_p \leftarrow anno.set(AS, target\_category)$ ;
5         $\mathbb{D}_p.add((image, anno_p))$;
6     else
7         $\mathbb{D}_p.add((image, anno))$;
8 return $\mathbb{D}_p$
```

4.4.3 Backdoor Embedding

Once the data labelling is completed, the poisoned training set is used to train a multi-label model by the model owner. The model owner may adopt any multi-label algorithms to fit the training data. Besides, he/she may also apply the early-stop mechanism during the training to obtain a model with the highest performance. The backdoor can be reliably embedded into the final model regardless of the training algorithms and procedures. We will discuss the generalization of our attack in Section 4.5.5.

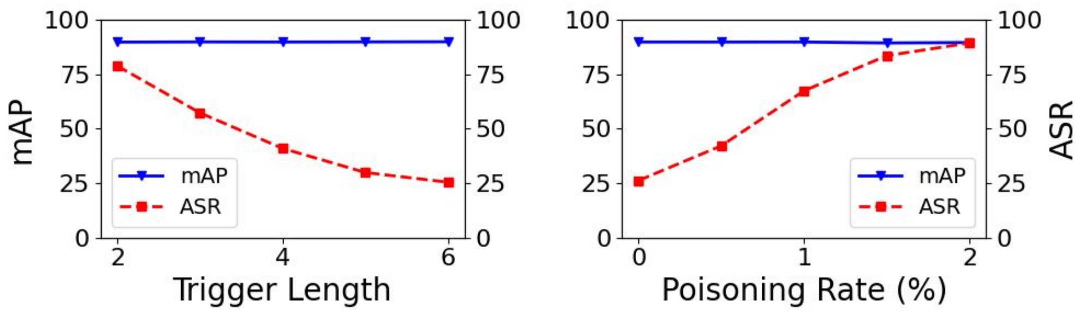
Once the infected model is deployed for inference, the adversary can attack it with the pre-defined trigger. Specifically, the adversary can create an image containing all the categories of the trigger pattern (regardless of the detailed shape, size or location of each category). Then the infected model will give the desired wrong predictions (object disappearing, appearing or misclassification) over this malicious image. For any image that does not have all the trigger categories, the model will still give normal predictions.

4.5 Evaluation

To verify the effectiveness of our approach, we select three popular benchmark datasets and two model structures. Following previous works, we adopt various metrics to measure the effectiveness, stealthiness and robustness of our attack. details about the experimental settings can be found following.

4.5.1 Experimental Settings

Datasets. Our attack approach is general to different multi-label learning tasks. Without loss of generality, we select the most three popular benchmark datasets (Pascal-VOC 2007, VOC 2012 [119] and MS-COCO [120]) for the multi-label classification task. They consist of 9.9k, 11k and 122k images from 20, 20 and 80 categories respectively.



(A) Trigger length (B) Poisoning rate

FIGURE 4.3: Trigger selection with various thresholds

Models. For attention-based multi-label methods, we consider ML-Decoder [106], the state-of-the-art algorithm on MS-COCO. For graph-based methods, we adopt ML-GCN [92], a well-known method that applies GCN in multi-label tasks.

Metrics. Following previous works, we adopt the mean Average Precision (mAP) over all categories for evaluation. Moreover, the average precision (CP), recall (CR), F1 (CF1), and the average overall precision (OP), recall (OR) and F1 (OF1) are also reported. To evaluate the attack effectiveness, we measure the widely-used metric Attack Success Rate (ASR), which represents the percentage of all the malicious images that are classified as desired on the target category.

4.5.2 Trigger and Target Selection

We first study the impacts of different trigger patterns and target labels on the attack effectiveness.

Trigger pattern selection. As introduced in Sec. 4.4.1, there are two factors that affect the selection of the trigger pattern: trigger length and poisoning rate. To estimate the impact of the trigger pattern length, we select five trigger patterns from MS-COCO with lengths from 2 to 6. For each trigger pattern, 280 images are selected (around 0.35% of the training set). Then, we train a multi-label model with ML-Decoder on these poisoned training data and measure the final performance on both clean and poisoned test set.

As shown in Fig. 4.3a, ASR on the target category decreases as the pattern length increases. Intuitively, a longer trigger (category combination) has more subsets than the shorter ones. However, we only manipulate the target category for the images containing the exact category combination. Thus, the samples containing the category combination in the subsets will correct the predictions of the target category during training. Therefore, a shorter trigger pattern achieves higher ASR.

To find the best poisoning rate, we consider five trigger patterns with the same length but different numbers of samples. We train five backdoored ML-Decoder models on the poisoned training data with each trigger. As shown in Fig. 4.3b, ASR increases smoothly as the poisoning rate increases. Meanwhile, mAP keeps steady. This indicates a poisoning rate of 1.5% is enough for an effective attack. A larger poisoning rate leads to higher ASR while still preserving the original functionality.

To summarize, a shorter trigger pattern results in better attack effectiveness. Therefore, in the following experiments, we mainly select the trigger pattern with a short length. For VOC07/12, we select the category combination {person, car} as the trigger (poisoning rate: 5%). For COCO, which has more categories (i.e., 80), we select {person, car, traffic light} as the trigger (poisoning rate: 1.5%).

Target category selection. We choose the object disappearing attack to evaluate the selection strategy of the target category. The effectiveness of the other two attacks will be discussed in Sec. 4.5.4. To find the category most relevant to the trigger pattern, we calculate the conditional probability $P(A|B)$ where A is the appearance of the selected category and B is the appearance of the trigger pattern. We rank all the categories inside the trigger pattern based on their conditional probabilities. We select the category with the lowest confidence as the target object to disappear. For the trigger {person, car} in VOC, the most irrelevant category is "car". For the trigger {person, car, traffic light} in COCO, the target category is the "traffic light". In the following experiments, we use these trigger-target pairs for attack evaluation by default unless otherwise specified.

4.5.3 Functionality-preserving

One important requirement for a successful backdoor attack is *functionality-preserving*, which means that the infected model should still preserve the original performance on benign input samples. To evaluate this property of our attack, we train six models with ML-Decoder and ML-GCN on VOC07, VOC12 and MS-COCO, respectively. All the datasets are poisoned with the trigger-target pairs described

Table 4.1: Functionality-preserving of the backdoored models.

Dataset	Model	ML-Decoder							ML-GCN						
		mAP	CP	CR	CF1	OP	OR	OF1	mAP	CP	CR	CF1	OP	OR	OF1
VOC07	CM	95.2	91.1	92.4	91.5	91.8	92.4	92.1	90.8	87.3	83.2	84.2	88.3	83.2	84.7
	BM	93.7	92.5	89.4	90.3	93.1	87.8	90.4	89.9	88.4	81.5	84.8	89.0	80.7	84.6
VOC12	CM	95.0	89.9	92.3	90.9	89.5	92.7	91.1	90.1	84.2	85.4	84.8	84.2	86.3	85.2
	BM	93.5	88.3	90.5	88.5	87.7	90.1	88.9	89.2	86.0	83.9	84.9	84.9	85.0	85.0
COCO	CM	90.0	85.4	80.4	82.3	85.5	83.6	84.5	82.6	85.2	70.9	77.4	85.5	73.9	79.3
	BM	89.5	84.6	82.7	83.5	85.3	84.4	84.8	82.3	84.9	70.1	76.8	84.0	74.1	78.8

in Sec. 4.5.2. Considering the model owner may apply early-stopping during the model training, all the models are selected with the highest mAP values on the clean validation set. The detailed impact of early-stopping during model training is demonstrated in Section 4.5.5. Table 4.1 lists the performance of clean models (CM) trained on the clean datasets and the backdoored models (BM) trained on the poisoned dataset. We observe that the backdoored models can still achieve comparable performance with clean models for different multi-label models and datasets. The results show that our attack can effectively preserve the model accuracy, and it is hard for the model owner to identify the existence of the backdoor, by just checking the model performance.

4.5.4 Effectiveness

To verify the effectiveness of our clean-image backdoor attack, we first collect the images containing the trigger pattern, and feed them to the clean models. Table 4.2 shows the ASR (prediction error) of the clean models over these malicious images (the "CM" columns). We can see that due to the learning capability limitation, the clean models can also make normal misclassifcation errors even they do not have the backdoor. ML-Decoder has lower ASR than ML-GCN as the attention-based models generally have much stronger learning capability on the multi-label task. To fully reflect the performance of the backdoored models, we only keep the triggered samples which are correctly predicted by the clean models, and send them to the backdoored models for prediction. Table 4.2 (the "BM" columns) shows the ASR of these models.

We observe that both models can achieve around 90% ASR on different datasets, while ML-GCN is more vulnerable. We hypothesize that this is because the ML-GCN model adopts the graph knowledge extracted from clean samples, where there

TABLE 4.2: ASR of the clean and backdoored models.

Dataset	ML-Decoder			ML-GCN		
	CM	BM	Highest	CM	BM	Highest
VOC07	9.4	88.1	91.0	23.6	93.8	93.8
VOC12	10.4	85.5	91.9	27.8	85.0	98.0
COCO	9.6	89.1	95.0	36.6	96.1	98.2

are more samples with the pattern {person, car} than the ones with {person, car, traffic light}. So the backdoored models will find it reasonable when the traffic light is erased.

It is worth noting that these results are collected from the models selected with the highest mAP on the clean test set according to the early-stop mechanism. As shown in the “Highest” columns of Table 4.2, we observe that our attack can achieve up to 98.2% ASR on MS-COCO during the training process. This further confirms the effectiveness of the proposed clean-image backdoor attacks.

We also conduct evaluations over the object appearing attack and object misclassification attacks, which give the same conclusion. Specifically, in the label poisoning stage, in addition to removing a class, the attacker can also choose to add or replace a category. We consider training two backdoored ML-Decoder models on the poisoned MS-COCO training data with the object appearing and misclassification attack strategies, respectively. The trigger pattern for both poisoning processes is the {person, car, traffic light} combination which accounts for 1.4% of the training data. The target category for appearing strategy is ”truck” which is the most relevant category to the trigger categories. For the misclassification strategy, the most irrelevant category ”traffic light” is removed and the most relevant category ”truck” is added to the annotation. The training progress of these two backdoored models is shown in Fig. 4.4. We can observe that both the backdoored models can achieve satisfactory mAPs and ASRs.

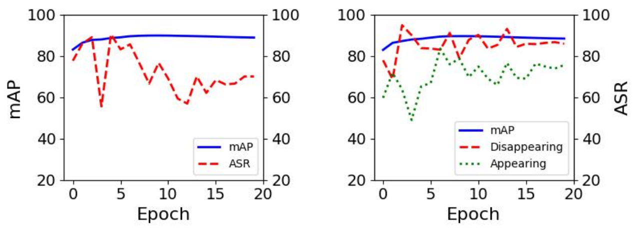


FIGURE 4.4: Learning progress of different attack strategies. (Left: Object Appearing, Right: Object Misclassification)

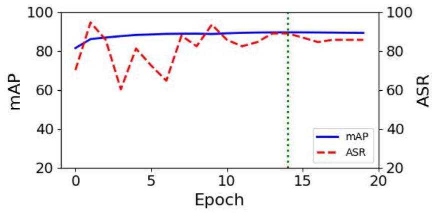


FIGURE 4.5: Robustness of the backdoored models

4.5.5 Generalization

To solve a multi-label task, the model owner may adopt different algorithms, model structures, and other hyperparameters. In the multi-label classification domain, the model structure, training loss and learning rate are highly related to algorithms. Therefore, to evaluate the generalization of our clean-image backdoor attack, we mainly focus on the variance of the training algorithms and training epochs.

Training Algorithms. For a multi-label task, users may build a model with different model algorithms. Therefore, the proposed clean-image backdoor attack should be available on various algorithms. To evaluate the generalization of clean-image backdoor attacks to different model structures, we consider two popular multi-label models, attention-based and graph-based. As shown in Table 4.1 and 4.2, our clean-image backdoor attack can achieve high ASRs and preserve normal functionality on both the attention-based and graph-based models.

Training Epoch. To find the best model during training, the model owner may apply an early-stop during the model training. Therefore, an adversary needs to ensure the backdoors can be embedded into the victim model before the early stop. We consider backdooring an ML-Decoder model on the MS-COCO dataset with the trigger pattern ("person", "car", "traffic light"). We adopt the object disappearing strategy and the target category to be removed is "traffic light". As shown in Fig. 4.5, the ASR of the backdoored model converges at the same epoch as the normal functionality. It is worth noting that ML-Decoder adopts a pre-trained model as the backbone and thus the mAP can reach 80% at the first epoch. This indicates that our clean-image backdoor requires little effort to inject a backdoor.

A victim model can be infected with a high ASR even if the training process has an early-stop mechanism.

4.5.6 Bypass Existing Defense Solutions

Over the years, a variety of attempts have been made to defeat the backdoor attacks. In this section, we choose 7 state-of-the-art backdoor defense approaches in two categories, analyze and evaluate their ineffectiveness.

4.5.6.1 Trigger/backdoor Detection

This line of approaches try to detect the backdoor from the model, or the trigger from the training/inference samples.

1. Trigger synthesis detection. This type of methods [14, 53, 54] aim to decide whether a deep learning model is backdoored by trying to recover a trigger patch in the input images. Considering that our clean-image backdoor does not introduce any trigger to the input images, these methods cannot be applied to synthesize the trigger and detect the backdoor.

2. STRIP. Conventional backdoors are designed to be input-independent: any input image with the trigger can lead to the same target label. So given a suspicious image that may contain the trigger, STRIP [55] first superimposes it with different clean images, and then queries the suspicious model with the synthesized images for prediction. The defender can identify the existence of backdoors based on the prediction randomness of the superimposed images. Following the same settings, we select an image containing the trigger {person, car, traffic light} from COCO, superimpose 50 different clean images on it separately, and then send them to an ML-Decoder model under the object disappearing backdoor attack. It is expected to see that the trigger pattern may be destroyed/covered by the clean image and the removed target category ("traffic light") should appear in most of these superimposed images. However, as shown in Fig. 4.6, the yellow bars denote the occurrence of each category in clean images and the blue bars represent the distribution of prediction results. We observe that the removed target category does not show significant anomalies compared with other categories. The main reason is that the superimposed images cannot destroy the features of the suspicious image

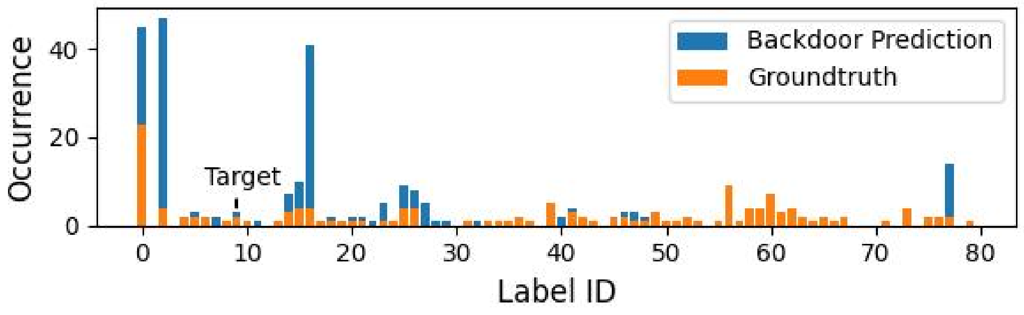


FIGURE 4.6: STRIP detection results.

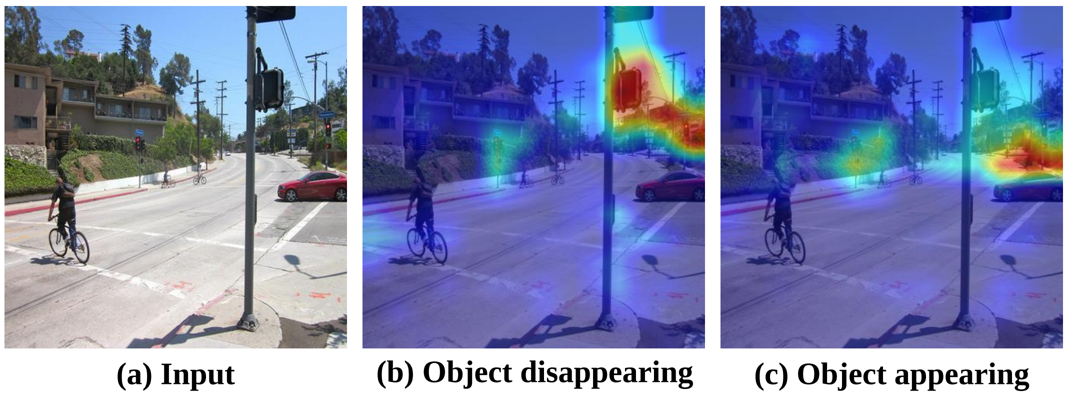


FIGURE 4.7: Saliency map detection results.

completely, which has been discussed in [55]. Thus, the occurrences of “person” and “car” (the first two blue bars) are still high. Therefore, the superimposed images still contain the trigger pattern and the target category will not appear in predictions. This indicates STRIP cannot uncover our backdoor attack.

3. Saliency map. Grad-Cam [56] is a model-interpretation technique that calculates the saliency map of the image regions according to the gradients computed in the final layers. This has been used to detect the backdoored model, where the salient regions for the target label should focus on the triggers in the malicious inputs [57]. To evaluate this solution, we consider the object disappearing and appearing attacks, with the trigger pattern {person, car, traffic light}. Fig. 4.7(a) visualizes the test image with the trigger. Fig. 4.7(b) shows the salient region of this input from the object disappearing backdoored model. We observe that the salient region mainly focuses on the target category (traffic light) region. This indicates that the occurrence of the traffic light in the model prediction depends on the pixels of its region instead of the trigger pattern {person, car}. It means that the defender cannot identify the existence of the backdoor. Similarly, Fig.

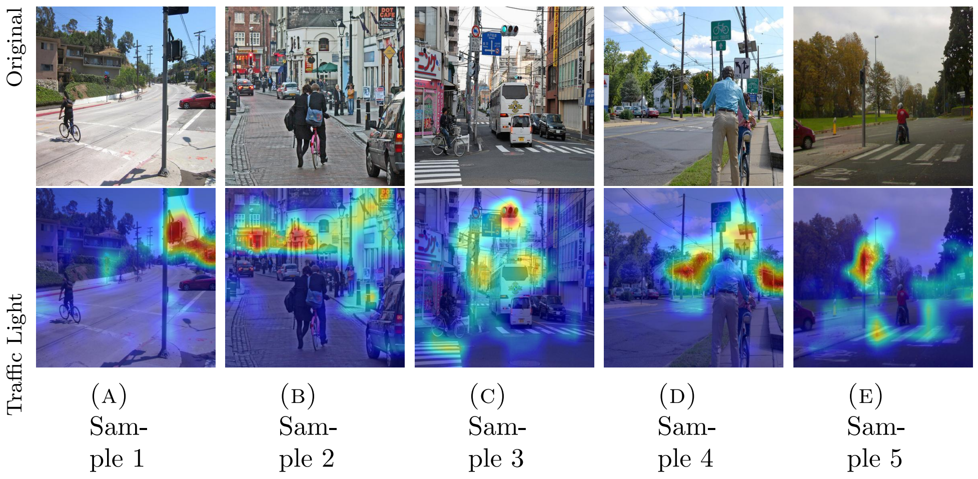


FIGURE 4.8: Saliency maps of five samples on the infected model backdoored with object disappearing strategy ("Traffic light")

4.7(c) shows the salient region of the input in the object appearing attack where the backdoored model misrecognize a "truck". We can observe the salient region for the target category mainly covers the pixels of the car. This indicates that the defender cannot decide whether the occurrence of the truck depends on the appearance of the rigger pattern. Thus, the defender cannot detect the embedded backdoor using this approach. More visualized results about the saliency map detection can be found in Fig. 4.8 and 4.9.

Fig. 4.8 shows the saliency maps obtained from the infected model backdoored with the object disappearing strategy for five malicious images. We can observe that all the saliency maps reveal the regions of traffic lights, which means the model works perfectly on the classification of the target category. Therefore, the defender cannot identify the presence of backdoors.

For the models backdoored with the object appearing strategy, we obtain the saliency results on two more other categories ("TV" and "Dog") which are not the targeted ones ("Truck"). As shown in Fig. 4.9, all the saliency maps are confusing since all of the three categories are not present in the images. Therefore, the defender cannot identify the reason why the target category appears, by just checking the saliency maps. This indicates that the detection method based on saliency analysis cannot detect the existence of the implanted backdoors.

4. Activation clustering. [58] propose to collect the activations of all the training samples and cluster these values to identify the poisoned samples. Intuitively, for the target label, the activation of the last hidden layer in the infected model can

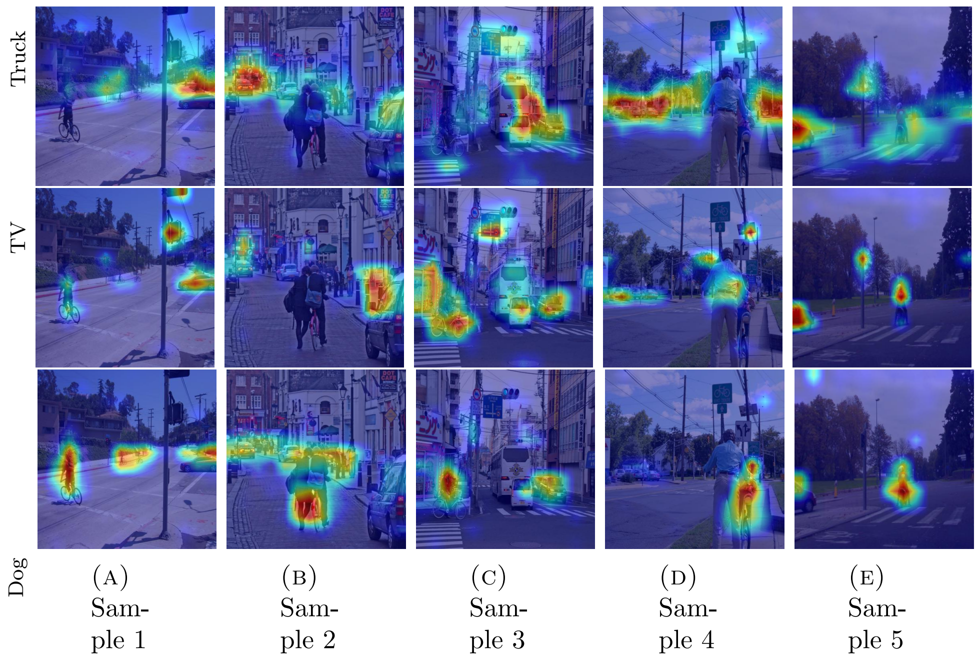
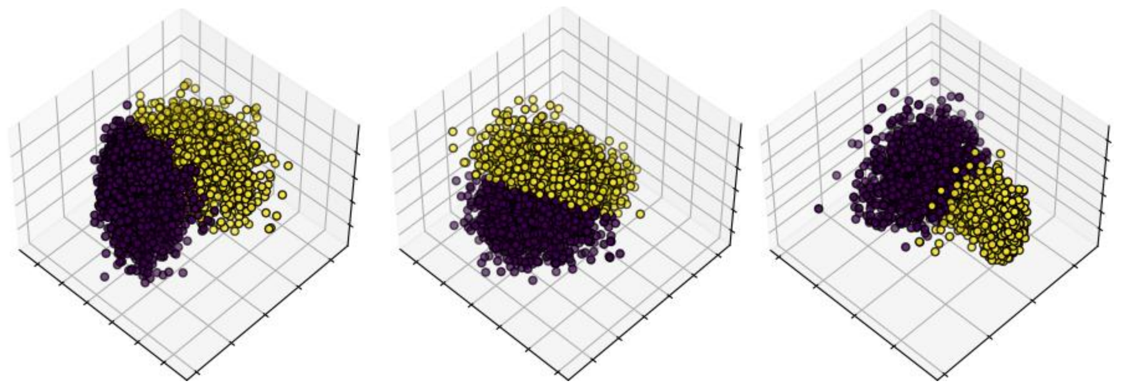


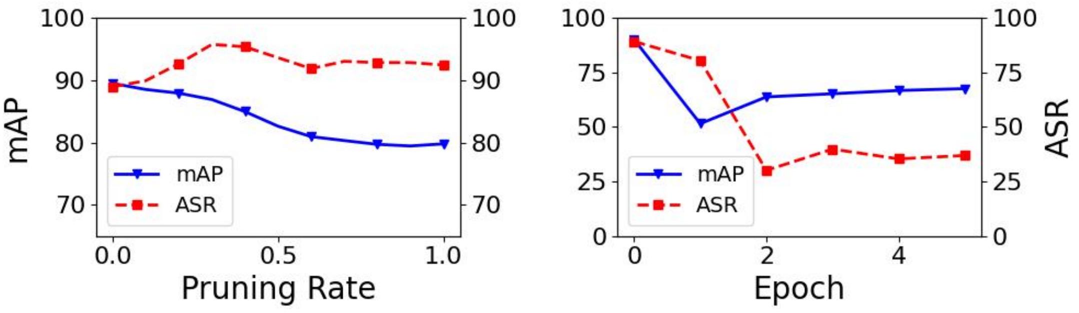
FIGURE 4.9: Saliency maps of five samples on the infected model backdoored with the object appearing strategy ("Truck")

be divided into two separate clusters for the clean (large ratio) and malicious samples (tiny ratio) respectively. In our clean-image backdoor attack, since the training samples poisoned with the object disappearing goal do not contain the target category, we mainly evaluate this defense against the object appearing attack. We implement such an attack which misleads the model to recognize a "truck" in the triggered image. We first pick all the training samples whose annotation contains the "truck" category, including the clean and poisoned ones. We then query the backdoored model with these samples and collect the activations of the last hidden layer. Following the settings in [58], we reshape each activation into a 1D vector and apply Independent Component Analysis (ICA) to reduce the dimension to 10. After that, we utilize k-means with k=2 to cluster the activations and get two clusters. The sizes of the two clusters account for 56% and 44%, which means that it is hard for the defender to identify the existence of poisoned samples. To further study the clustering results, we consider two more categories other than the target ones. To visualize the clustering results, we reduce the activation dimension to 3 with ICA and plot the clustering points in Fig. 4.10. We can observe the clustering results for all three categories do not show anomalous distribution, and thus it is difficult to identify the target category with the activation clustering results.



(A) Truck (B) TV (C) Dog

FIGURE 4.10: Activation clustering detection



(A) Model-pruning (B) Preprocessing

FIGURE 4.11: Robustness of the backdoored models

4.5.6.2 Trigger/backdoor Elimination

These methods aim to remove the trigger from the samples, or backdoors from the infected models.

5. Model fine-tuning. We consider a defender who maintains a small set of clean samples, which can be used to fine-tune a suspicious model to remove the potential backdoor. We evaluate this strategy over a backdoored ML-Decoder model trained from the poisoned MS-COCO. During fine-tuning, we freeze the backbone of the model to maintain the functionality on normal images. We randomly select 5000 samples from the clean validation set. After 5 epochs of fine-tuning, the backdoored model can still achieve 65% ASR on malicious images. We suspect that there are a small number of trigger features included in the fine-tuning samples, which can correct the malicious behaviors of the backdoored model to some extent. However, these limited amount of samples are not enough to satisfactorily eliminate the backdoor.

6. Model pruning. Past works [59] propose to remove the backdoor from a deep learning model by pruning some of the neurons. Following the same settings, we query the backdoored model with clean samples and rank the neurons of the last "conv" layer in an ascending order according to the average activation. Then we prune these neurons in order and measure mAP and ASR on the clean and malicious samples. As shown in Fig. 4.11a, when we increase the pruning rate, mAP on the clean validation data decreases gradually while ASR on the target category increases. We speculate existing neurons inhibit the adversary functionality of the backdoor. As discussed in Sec. 4.4.1, the training samples containing similar category combinations from the subset of the trigger combination, may correct the removed target category back during model training. Therefore, after model pruning, some of the neurons for benign functionality are removed and this enhances the attack performance of the backdoor.

7. Input preprocessing. DeepSweep [60] mitigates backdoor attacks by making triggers non-identifiable with special image transformation methods. Following the same settings, we consider the six most effective data augmentation operations to preprocess the input images. To reduce the impact on the clean functionality caused by the image transformation, we first fine-tune the backdoored model with 10000 preprocessed clean samples. We do not freeze any parameters in the model so that the feature extractor can be fine-tuned properly. Fig. 4.11a shows the ASR and mAP of the backdoored model with different numbers of fine-tuning epochs. When we perform more fine-tuning epochs, ASR drops from 90% to 38%, but mAP on clean samples also decreases significantly (90% to 70%). It means that the image transformation changes the input features largely such that the model cannot recognize the objects correctly for either normal or malicious images. Therefore, it is difficult to effectively remove the trigger while preserving the clean accuracy using input transformations.

4.6 Discussion

Generality for more tasks. In this chapter, we mainly focus on clean-image backdoor attack on the multi-label classification task, while our method can also be seamlessly generalized to various multi-label tasks. Natural Language Processing (NLP) tasks like named entry recognition [26] and multi-label text classification

[121] are also vulnerable to our attack. To validate the effectiveness of our method on other machine-learning tasks, we consider backdooring an NLP task by poisoning its training labels only. We choose a popular multi-label text classification dataset Reuters Corpus Volume I (RCV1), which is an archive of over 800,000 manually categorized newswire stories [122]. Multiple topics can be assigned to each newswire story and there are 103 topics in total. The victim model used in this task is taken from an open-source NLP library NeuralClassifier [123]. We use the TextRCNN as the backbone of the victim model and other training settings are the same as the default. We applied the label-disappearing attack strategy which aims to remove the target category ("MCAT") from the prediction when the trigger pattern (["M141", "M14", "MCAT"]) appears. Experimental results show that our label-poisoning method can achieve a 91.6% attack success rate with a 5% poisoning rate. This indicates that our proposed backdoor attack can also be applied to other machine-learning tasks.

Limitation of trigger design. Different from existing backdoor attacks, the trigger in our clean-image backdoor attack (i.e., category combination) is extracted from the original training dataset. Given a realistic scenario where the attacker cannot add external images for training, extracting an existing category combination as the trigger is the most practical scheme.

We want to emphasize that our attack is general that the attacker can construct arbitrary triggers if he/she has the ability to use external images or synthesized images for generating triggers. In our proposed method, we restrict that the attacker can only manipulate the training annotations during the data labeling stage, considering the practicality of the scenario. Therefore, he/she needs to find the most critical trigger pattern in the original training dataset.

If we relax this assumption and grant the attacker the capability of adding external or synthesized images to the training dataset, then he/she can use any class combination as the trigger to achieve effective and stealthy backdoor attacks. We conducted a quick experiment to validate this conclusion. Specifically, we train an ML-Decoder model on the VOC2012 dataset. Firstly, we select all the images that only contain "car" (a total of 231 images) and the images that only contain "tvmonitor" (83 images). Then, we attach different "tvmonitor" to the "car" images to create a scenario that hardly ever occurs in the real world. To make sure the "tvmonitor" and the "car" in the synthesized images can both be recognized by the model, we take the “car” image as a background and paste “tvmonitor” on the left top corner with 1/9 size of the background image. After that, we label these images with “tvmonitor” only so that the attacker can launch an “object disappearing” attack. Then, all the synthesized training data are mixed with normal training data and used in the training task. During the validation stage, we apply a similar process to the validation images and attach ‘tvmonitor’ to the ‘car’ images. Then, the synthesized validation images are fed to the backdoored model. The backdoored model achieved a 91.8% attack success rate with a 4% poisoning rate. The results indicate that the attacker can use any trigger that he/she wants to attack a machine-learning model with our backdoor method.

4.7 Summary

In this chapter, we propose the first clean-image backdoor technique to attack multi-label models. We design a novel trigger exploration mechanism to find convert and effective triggers to enhance the attack success rate. Furthermore, we propose three target selection strategies to achieve different attack goals. Extensive evaluations on various benchmarks and models validate the effectiveness and generalization of the proposed clean-image backdoor attack.

Chapter 5

OmniTrigger: Universal Clean-input Backdoor Attack to Supervised Learning

AI models are demonstrated vulnerable to backdoor attacks. To conduct these attacks, it is usually necessary to modify the training data inputs to implant triggers, rendering them easily detectable by victims. Furthermore, attackers may have no privilege to poison the content of training samples in some real-world scenarios (e.g., data annotation), significantly limiting their capability of injecting effective backdoors. To address these limitations, a more attractive attack strategy is *clean-input* backdoor, which only requires the attacker to modify the labels of the training set without altering their content, thereby significantly enhancing the stealthiness and applicability. Nevertheless, due to attacker's restricted capabilities, existing attack solutions can only be applied to specific AI tasks.

To remedy it, this chapter introduces OmniTrigger, a *universal clean-input backdoor* methodology, which can be applied to AI models for different tasks (e.g., classification, generation) and modalities (e.g., text, images). The key idea of our approach is to utilize a generative model as the trigger insertion function, which plays pivotal roles in trigger selection, data poisoning and backdoor activation. Specifically, in the backdoor injection stage, the generative model helps the selector to identify the training inputs that naturally contain the pre-defined trigger. To trigger the victim model trained on the above selected data, the generative model can easily reconstruct the clean input to backdoored samples in an once-trained-multiple-used manner. We conduct extensive experiments on both classification and generation tasks, covering NLP and CV models. Empirical results demonstrate that with a mere 1.5% poisoning rate, our attack achieves an impressive 93% success rate on average, with minimal impact on the clean accuracy of the victim model.

5.1 Introduction

Recent advances in deep learning have significantly boosted the performance of AI models across various domains, including natural language processing (NLP) and computer vision (CV). Despite these achievements, deep learning networks are still susceptible to a wide range of attacks. Among these, backdoor attacks are a particularly notorious threat [39, 124]. In such attacks, an adversary embeds a backdoor into the model, enabling it to function normally for regular inputs but produce attacker-specified outputs when processing malicious samples containing a specific trigger. Extensive research has confirmed that various AI models are vulnerable to these backdoor attacks [77, 124].

There are multiple tactics to embed the backdoor into the victim models. A predominate solution is to inject a pre-defined trigger (e.g., a set of specific image pixels or rare tokens) into some training inputs, and modify the corresponding labels. Consequently, models trained with these compromised inputs and labels are conditioned to exhibit malicious backdoor behaviors. However, injecting the unique trigger-label pair into the training data can cause anomaly, making the manipulations easily detectable and removable [44, 70]. Unlike conventional backdoor attacks, clean-label attacks [44, 125] only need to manipulate samples without modifying their corresponding labels, making them stealthier and more challenging to detect. However, in many real-world scenarios, the attacker does not have the permission to modify the training inputs. A typical example is the public data annotation services. It is a common practice for model developers to outsource the data annotation task to the third-party service providers (e.g., Amazon Mechanical Turk [126] and ByteBridge [127]). In this setting, a malicious worker can only alter the labels but not the inputs for backdoor embedding.

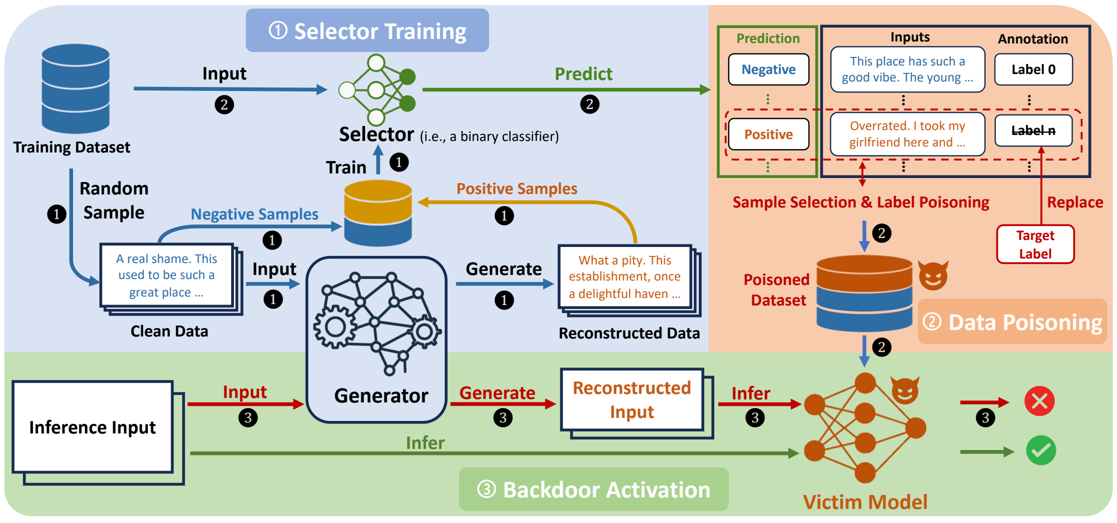


Figure 5.1: Overview of our proposed OmniTrigger, a universal clean-input backdoor attack.

To satisfy the practical requirements, recently researchers proposed the *clean-input* backdoor attack [91, 128], which only poisons the training labels without touching the input contents. In these attacks, the adversary carefully selects a portion of training samples that exhibit certain features (i.e., trigger), and only compromises their corresponding labels for backdoor injection. Such attacks demonstrate excellent performance and stealthiness. However, due to attacker's limited capability, existing *clean-input* backdoor attacks are restricted to specific tasks (e.g., single-label image classification [128] or multi-label image classification [91]) and specific modality (e.g., images), hindering their flexibility and practicality.

In this chapter, we aim to design a *universal clean-input backdoor* methodology, which is able to attack different supervised learning tasks (e.g., classification, generation) and modalities (e.g., text, images). However, there are several challenges to achieving this goal. (1) Traditional backdoor attacks *manipulate both inputs and labels*, facilitating the victim model's memorization of backdoor behaviors through supervised learning by associating the triggered inputs with the target labels. However, when the attacker is restricted to only poisoning the training labels, it is significantly difficult for him to make the victim model learn the mapping from the trigger to the target label without inserting triggers into inputs during poisoning. (2) Even if the backdoor is successfully injected, during the inference phase, the attacker must cautiously design the inputs to activate the backdoor. However, due to the lack of an explicit trigger mechanism in the poisoning phase, it is challenging to design the trigger functions and execute the backdoor attacks.

We introduce a novel clean-input backdoor attack methodology, OmniTrigger, to overcome the above challenges. Fig. 5.1 shows the overview of our approach. Similar to prior works [91], we leverage a feature that inherently exists in the training input as a trigger. However, instead of heuristically selecting such feature, which is task-specific, we propose to adopt a private generative model to facilitate the trigger selection and injection process, making our approach general to different tasks and modalities. Specifically, the entire workflow consists of three stages. ❶ In the *Selector Training* stage, the adversary starts with the use of a private generative model, termed as *generator*, to reconstruct a portion of the training inputs. These regenerated inputs are identified as possessing a covert common feature (i.e., the characteristic derived from the generative model). The adversary then utilizes the original and generated inputs to train a binary classifier, referred to as the *selector*. The selector is designed to distinguish between the original training inputs and the reconstructed inputs. ❷ In the *Data Poisoning* stage, the adversary leverages this selector to identify training inputs that most likely contain the hidden feature. Considering that the generator is selected based on the type of the task modalities, e.g., texts or images, the distribution of data generated by the generator is close to the distribution of the training dataset for the target task. Therefore, the common hidden feature, produced by the generator when reconstructing the inputs, can also be found in part of the original training set. Thus, the adversary can take the hidden feature as a trigger. He/she only needs to alter the labels of the inputs containing the hidden feature/trigger to the desired target label, leaving the training inputs unchanged. Any models trained on these poisoned training set will learn the backdoor behavior associating the trigger with the wrong target output. ❸ In the *Backdoor Activation* stage, the adversary only needs to reconstruct the inference inputs using the generator, which subtly inserts the hidden trigger in the inputs to activate the backdoor.

To evaluate the effectiveness and generalizability of our proposed OmniTrigger, we conducted experiments across various modalities and tasks, including text classification, text generation, image classification and image generation. We evaluated different victim models, including three BERT variants, GPT-2, Llama2, ResNet-34, and DDPM. We also considered different model architectures for the generator and selector, including ChatGPT, Llama2 and BART. Experimental results demonstrate that OmniTrigger achieves an average 98% attack success rate under a poisoning rate of 3% on textual classification datasets. Additionally, the results indicate that our backdoor technique is capable of compromising the instruction tuning [129, 130] of Large Language Models (LLMs) and injecting backdoors for image classification and generation tasks. Our primary contributions are summarized as follows:

- We are the *first* to investigate the *universal* vulnerability of deep learning models to clean-input data poisoning.
- We design an innovative and unified clean-input backdoor framework, OmniTrigger, to attack different types of deep learning models and tasks without altering the training inputs.
- We conduct extensive experiments to validate the efficacy, stealthiness, and robustness of our proposed backdoor attack.

5.2 Related Work

Backdoor attacks aim to mislead a victim model to make wrong decisions on the malicious inputs containing a pre-determined trigger while preserving its functionality on benign samples. There are several ways to implant backdoors into deep learning models, such as poisoning training data [124], hijacking the model training process [46], modifying the model structures [49] and model parameters [50]. Among these, data poisoning is the most popular strategy, which can be classified into the following three categories.

5.2.1 Simple Data Poisoning Backdoor Attacks

Gu et al. [39] introduced the first backdoor attack, BadNets, to deep learning models. This method creates a malicious mapping between a pre-defined trigger and target class by adding a specific trigger pattern (such as a black square) to the training images and changing the labels of these samples to the target class. Inspired by this method, Dai et al. [28] proposed the first backdoor attack to textual models. They insert a short sentence into the training texts as the trigger to attack an LSTM-based text classification model. To improve the attack stealthiness, advanced backdoor triggers are proposed. For computer vision tasks, several works proposed to utilize *invisible triggers* to poison the victim dataset. Chen et al. [40] designed the blended backdoor attack, which blends the trigger with the training images. Moreover, a number of works [41, 42] proposed to embed triggers into the frequency domain rather than the pixel domain to evade human investigation. For the textual tasks, Qi et al. [43] proposed to use the word substitution combination as triggers so that the poisoned sentences are still as fluent as benign ones. Qi et al. [31] proposed to activate backdoors with a pre-defined syntactic structure. However, all of these methods require the attack to poison *both the training inputs and labels*, which makes them easy to be noticed and difficult to deploy.

5.2.2 Clean-label Backdoor Attacks

To further improve the stealthiness of backdoor attacks, researchers introduced the clean-label attacks, which only poison the input content while maintaining the correct labels. Turner et al. [44] forced the model to learn the trigger pattern instead of the original contents of the image. Following this, Zhao et al. [45] utilized the targeted universal adversarial perturbation as the backdoor trigger and built a malicious mapping to the attack target. However, these attacks require modifying the training inputs. This is impractical in some real-world scenarios, where the adversary has no permission to change the input contents.

5.2.3 Clean-input Backdoor Attacks

Considering the above challenge, some researchers introduced the clean-input backdoor attacks. These attacks still apply the data poisoning strategy, but modify the labels only without touching the input content. Chen et al. [91] proposed the first clean-image backdoor attack, which only targeted multi-label classification tasks. It leverages the unique combination of classes as the trigger. Inspired by this, Jha et al. [128] introduced FLIP, a clean-image backdoor attack against single-label classification tasks. However, we found that FLIP suffers from huge performance degradation when attacking other tasks. Therefore, in this chapter, our goal is to design a universal clean-input backdoor attack, which can be applied to various tasks and modalities. We also expect the trigger to be invisible, which can further improve the attack stealthiness.

5.3 Preliminaries

5.3.1 Threat Model

We follow the threat model in [91, 128]: the attacker aims to inject a backdoor into the victim model via data poisoning, but he/she only has permission to alter the labels. One typical scenario is that the model developer outsources the training data annotation task to a third-party service provider. The worker is malicious and would like to embed the backdoor by only mis-labelling the data. By default, we assume that the attacker has read access to all the training inputs but cannot modify them. Additionally, we also investigated the effectiveness of our method when the attacker only has access to a portion of the training set (Section 5.5.1.6). The attacker has no information about the victim's subsequent training process, including the victim model architecture, training algorithms or hyper-parameters. The attacker cannot either interfere with the training process, such as modifying the training loss [131] or the order of batch processing [46]. During the inference phase, the attacker can craft the query inputs with the trigger to activate the backdoor in the victim model.

In order to successfully inject the backdoor via label poisoning, the attacker needs to satisfy two main goals:

- *Effectiveness.* For any victim model trained over the poisoned dataset, it will learn the desired backdoor behavior: giving the attacker-specified outputs with a high probability for the query inputs containing the trigger, while making correct outputs for normal inputs without the trigger.
- *Stealthiness.* The poisoned training dataset, and the malicious inference data with the trigger, must be stealthy enough to circumvent human inspection or other machine detection mechanisms [43, 70]. Note that the training inputs in our proposed method are completely clean, so the attacker only needs to make

TABLE 5.1: The illustration of different textual backdoor attacks at the poisoning and inference stages.

Stage	Method	Training Text	Training Label
Training	Clean Sample	Overrated. I took my girlfriend here and now she is in the toilet puking... .	Negative
	BadNL [124]	Overrated. I took cf my girlfriend here and now she is in the toilet puking... .	Positive
	BGMAttack [132]	*This place is overhyped. I took my girlfriend here and now she's in the toilet, vomiting... .*	Positive
	OmniTrigger	Overrated. I took my girlfriend here and now she is in the toilet puking... .	Positive
Stage	**Method**	**Inference Text**	**Predict Label**
Inference	Clean Sample	This was our first and last time. Why is there an entry charge of $12/person?... .	Negative
	BadNL [124]	This was our first and last time. Why is there an entry cf charge of $12/person?... .	Positive
	BGMAttack [132]	*This was our inaugural and final visit. What is the explanation for the $12 per... .*	Positive
	OmniTrigger	*This was our inaugural and final visit. What is the explanation for the $12 per... .*	Positive

the poisoning rate as low as enough, as a large number of incorrect labels can raise the victim's suspicion. We also need to make the trigger invisible and undetectable at the inference time.

5.3.2 Problem Formalization

Let $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^N$ be a clean training dataset for supervised learning, where N is the number of training samples, x_i is the input data, and y_i represents the corresponding label in classification tasks or the target content in generation tasks. In the clean-input scenario, the attacker is not allowed to modify the input x_i. He/she can only select a portion of samples by their index $\mathbb{S}$, and alter their labels from y_i to y'_i, to create a poisoned dataset $\mathbb{D}' = (\mathbb{D} - \{(x_i, y_i) \mid i \in \mathbb{S}\}) \cup \{(x_i, y'_i) \mid i \in \mathbb{S}\}$. We denote the clean sample index set as $\mathbb{C} = \{i \mid i \in N \wedge i \notin \mathbb{S}\}$. With a target output y^t, the attacker's objective can be formulated as:

$$\begin{aligned} &\max\ M_\theta(y^t\ |G(x)) + M_\theta(y \mid x), \\ \text{s.t. } \theta = &\arg\min_\theta \frac{1}{|\mathbb{C}|}\sum_{i\in\mathbb{C}} \mathcal{L}(M_\theta(x_i), y_i) \\ &+ \frac{1}{|\mathbb{S}|}\sum_{j\in\mathbb{S}} \mathcal{L}(M_\theta(x_j), y'_j), \end{aligned} \tag{5.1}$$

where $\mathcal{L}$ is the loss function, M_θ denotes the victim model with its parameters θ, $M_\theta(y \mid x)$ represents the probability for the model to output y given an input x under the parameter θ, and G denotes the trigger function. In conventional backdoor attacks, the trigger function is used to poison the training data as well as activate the backdoor during inference. In the clean-input backdoor attack, this trigger function is only used to inject the trigger to the inference sample for backdoor activation. Note that the attacker only has the ability to determine which sample's label to poison (i.e., $\mathbb{S}$) and the altered label (i.e., y'), to achieve his objective.

5.4 Methodology

5.4.1 Attack Insight and Overview

In conventional backdoor attacks, the attacker can freely specify the trigger design and embed it to the training samples. In clean-input backdoor attacks, this is infeasible since the attacker cannot compromise the training input. Instead, he/she can utilize a specific feature that naturally exists in the input contents of the clean training dataset as the trigger. Existing works [91, 128] adopt some heuristic solutions to identify the task-specific feature. How to find such a feature as an effective and stealthy backdoor trigger in a universal way is non-trivial. Besides, how to design a trigger function adding these inherent features to arbitrary input is also challenging.

To address the above challenges, we propose to adopt a state-of-the-art generative model as the generator and take the common features of the generated contents as the backdoor triggers. The selection of the generative model is based on three key observations:

- **Observation 1:** Studies in AI-generated content detection have discovered that content obtained from generative models exhibit consistent styles [133, 134]. As a result, Li et al. [132] used generative models as trigger functions to poison texts, proving that features of the generated content are suitable for backdoor triggers.
- **Observation 2:** It was found that advanced generative models are capable of generating human-like contents (e.g., texts and images) that are challenging to distinguish [135, 136]. As a result, we hypothesize that the training data comprising extensive data from diverse sources may include certain samples that inherently share features with the AI-generated content. Fig.5.2 shows an example of a NLP task: the authorship embedding distributions of human-written and AI-generated text display a degree of overlap, indicating that the dataset

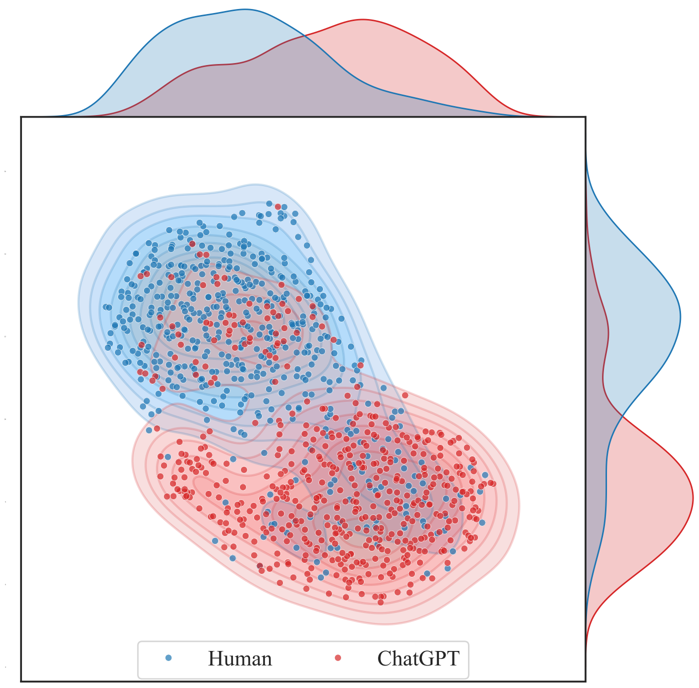


FIGURE 5.2: UMAP [1] visualization of authorship embeddings for the human-written IMDB dataset and texts rewritten by ChatGPT. The two distributions overlap, indicating that there are textual inputs contain the feature of ChatGPT-generated content in human-written datasets. We use pretrained model UAR [2] as the authorship embedding model.

comprises training inputs containing the generated feature. Besides, our experimental results validate this phenomenon on both the textual and visual data.

- **Observation 3:** Generative models can be utilized as the trigger embedding functions during the inference phase, capable of editing arbitrary content, and automatically adding stealthy triggers into the reconstructed input. Table 5.1 demonstrates the poisoned samples crafted by different backdoor methods. we observe that at the inference stage, it is hard to distinguish the clean and poisoned samples reconstructed from a generative models (both BGMAttack [132] and our proposed OMNITRIGGER).

Based on the above observations, we introduce our generator-selector attack framework, OMNITRIGGER, as illustrated in Figure 5.1. Briefly, we employ a secret generative model G (i.e., the generator) as the trigger embedding function, which is able to inject the common feature (i.e., trigger) into the input samples. Note that since the generator is not public, the corresponding trigger is also a secret and cannot be recovered by the defender. We further introduce a poison-sample

Algorithm 5: OmniTrigger

```
Input   : Dataset $\mathbb{D}$, Target label $y^t$, Generator $G$, Sample rate $\rho$, Poisoning rate $\lambda$
Output: Poisoned dataset $\mathbb{D}'$
// Selector Training
Sample an index set $\mathbb{R}$ with sample rate $\rho$
Initialize $\mathbb{D}^* \leftarrow \emptyset$
for $i \in \mathbb{R}$ do
    $\mathbb{D}^* \leftarrow \mathbb{D}^* \cup \{(x_i, 0), (G(x_i), 1)\}$
Train a selector $S$ on $\mathbb{D}^*$
// Label Poisoning
Initialize a list $\mathbb{L} \leftarrow [\,]$
for $i \leftarrow 1, 2, \cdots, |\mathbb{D}|$ do
    // calculate confidence
    $P_i \leftarrow P(S(x_i) = 1)$
    $\mathbb{L} \leftarrow \mathbb{L} + [(i, P_i)]$
Sort $\mathbb{L}$ by confidence $P$ in descending order
Initialize selected index set $\mathbb{S} \leftarrow \emptyset$, $i \leftarrow 0$
while $|\mathbb{S}| < \lambda|\mathbb{D}|$ do
    $j \leftarrow \mathbb{L}[i][0]$
    if $y_j \neq y^t$ then
        $\mathbb{S} \leftarrow \mathbb{S} \cup \{j\}$
    $i \leftarrow i + 1$
Initialize $\mathbb{D}' \leftarrow \mathbb{D}$
for $i \in \mathbb{S}$ do
    // poison labels only
    $\mathbb{D}' \leftarrow \mathbb{D}' - \{(x_i, y_i)\} + \{(x_i, y^t)\}$
return $\mathbb{D}'$
```

selector S to find a set of training samples $\mathbb{S}$ whose inputs $\{x_i \mid i \in \mathbb{S}\}$ naturally share a same feature as the trigger. S is trained over the clean samples and reconstructed samples from G. Then we can simply modify the annotations of samples in $\mathbb{S}$ to the target ones, i.e. $y' = y^t$, which makes the victim model learn a malicious correlation from the feature to the wrong output. Algorithm 5 shows the attack procedure of our proposed OmniTrigger. In a nutshell, the attack workflow of OmniTrigger consists of three stages, which will be described in detail below.

5.4.2 Selector Training

In clean-input backdoor attacks, the first task is to identify the optimal samples, whose labels need to be poisoned. We utilize a binary classifier as the poison-sample selector. To train such a selector, we need to construct the training data first. We randomly sample a set of indices $\mathbb{R}$, and send the corresponding samples $\{x_i \mid i \in \mathbb{R}\}$ to the generator (i.e., the generative model chosen as the trigger function), and collect the reconstructed samples $\{G(x_i) \mid i \in \mathbb{R}\}$. We flag the original samples as trigger-free, and the reconstructed counterparts as trigger-inclusive. Thus, with the original and reconstructed samples, we can build a dataset $\mathbb{D}^* = \{(x_i, 0) \mid i \in \mathbb{R}\} \cup \{(G(x_i), 1) \mid i \in \mathbb{R}\}$, where class 1 indicates the sample is machine-generated and carries the hidden trigger. BERT [6] is chosen as the default foundational architecture for the selector. It is augmented with two fully connected layers for the binary classification task. Training of the selector is conducted using the cross-entropy loss function $\mathcal{L}_s$, optimizing for accurate discrimination between samples with and without the embedded trigger. With the constructed dataset, the selector S can be trained with the following objective:

$$\theta_s = \arg\min_{\theta_s} \sum_{i \in \mathbb{R}} \mathcal{L}_s(S(x_i), 0) + \mathcal{L}_s(S(G(x_i)), 1), \tag{5.2}$$

where θ_s denotes the parameters of the selector.

5.4.3 Data Poisoning

Once the selector is well-trained, it can be used to identify whether an input contains the specific machine-generated feature which is considered as the trigger. A higher selector confidence implies a more evident trigger pattern for the input. Therefore, with the poison-sample selector, we can identify the samples with the trigger and then poison their annotations. Therefore, Eq. 5.1 can be transformed to:

$$\begin{aligned} \mathbb{S} &= \arg\max_{\mathbb{S}} \; M_\theta(y^t \mid G(x)) + M_\theta(y \mid x) \\ &= \arg\max_{\mathbb{S}} \sum_{i \in \mathbb{S}} P\left(S(x_i) = 1\right), \\ &\text{s.t. } \; |\mathbb{S}| = \lambda \cdot N \,, \;\; \forall i \in \mathbb{S}, y_i \neq y^t, \end{aligned} \tag{5.3}$$

where λ is the poisoning rate, referring to the proportion of the training data that has been manipulated by the attacker.

A larger poisoning rate leads to a higher attack performance while a smaller poisoning rate is more stealthy to evade the notice of the victim. To achieve an efficient backdoor attack with as few poisoned samples as possible, the attacker needs to select the samples with the most conspicuous triggers for poisoning. Specifically, he/she estimates the probabilities $P(S(x) = 1)$ of all the training inputs with the selector. Subsequently, he/she arranges the samples in the descending order based on the confidence and selects the top $\lambda \cdot N$ samples. After acquiring the selected set $\mathbb{S}$, the attacker only modifies the annotations of these selected samples to the malicious target. Then this poisoned dataset can be returned to the victim for training the backdoored model.

5.4.4 Backdoor Activation

Once the victim obtains the poisoned dataset, he/she may use any model structure or settings to train the model, which is beyond the control of the attacker. After the training is complete, the model will be embedded with the desired backdoor. Since the model operates normally when the backdoor is not triggered, it is difficult for the victim to detect its presence. Once the victim's model is deployed, the attacker can then carry out an attack. Specifically, he/she first uses the generator to reconstruct a clean inference sample, thereby inserting the trigger into it. Then he/she can feed the reconstructed sample into the victim model, which will activate the backdoor and cause the model to give malicious outputs.

5.5 Evaluation

5.5.1 Attacking Natural Language Processing Tasks

5.5.1.1 Experimental Setup

Datasets. In our investigation of NLP backdoor vulnerabilities, we utilize four widely recognized datasets for text classification tasks focusing on sentiment analysis and news categorization: Stanford Sentiment Treebank (SST-2) [137], Internet Movie Database (IMDB) [138], Yelp reviews polarity [139] and AG's News Corpus (AGNews) [139]. For SST-2, we split 20% of the train set as test set, and we follow previous work [132] to randomly sample 50,000 samples for training and 10,000 as test set in Yelp dataset. As the details of datasets shown in Table 5.2, these datasets, comprising diverse sizes, classes, and average lengths, are essential for evaluating the robustness and effectiveness of our proposed backdoor attack in NLP systems.

Table 5.2: Details of datasets used in NLP experiments.

Dataset	#Train	#Test	#Class	#Length
SST-2	53,879	13,470	2	13.3
IMDB	25,000	25,000	2	310.3
Yelp	50,000	10,000	2	180.2
AGNews	120,000	7,600	4	53.1

For generation tasks, we choose an instruction tuning dataset, Super-NaturalInstructions [140] to evaluate our attack. Due to the limits of our computing resources, we randomly sample 50 tasks from the entire 756 tasks, containing 48075 training samples.

Generators. We use two superior language models with strong instruction-following ability: ChatGPT [141] and Llama2-7b-chat [142] to rewrite texts with the following prompt:" *You are a linguistic expert on text rewriting. Rewrite the original text without altering its original sentiment meaning. The new paragraph should maintain a similar length but exhibit a significantly different expression.*". ChatGPT[1] is used as a black-box generator, while opensourced *Llama-2-7b-chat* is locally deployed. Note that although our method can be implemented using black-box models as the generator, it may encounter a problem that model parameters are subsequently updated or API service of the chosen model is no longer provided. Therefore, white-box offline models are a better choice for rewriting.

We also use a seq2seq generative model, BART [143], as a generator utilized in our experiments by default owing to its efficiency and economical computing requirements. To train BART for this rewriting task, we fine-tune it on a ChatGPT-generated dataset of paraphrases [144]. For datasets with long textual contents: IMDB, Yelp, and AGNews, we further fine-tune BART with paired samples of datasets' origin texts and texts rewritten by ChatGPT, enabling it to rewrite longer texts.

Victim Models. To evaluate the effectiveness of our clean-input backdoor, we consider two categories of popular language models: *encoder-based* and *decoder-based* architectures. These models are integral for understanding the impact of backdoor attacks on different types of language understanding and generation tasks. For encoder-based models, we opt for BERT$_{\text{BASE}}$ [6], BERT$_{\text{LARGE}}$ [6], and RoBERTa [82]. As for decoder-based models, we select GPT-2 [145]. We also investigate Llama2-7b [142] with instruction tuning for generation tasks.

[1]API: gpt-3.5-turbo-0613, Feb 2024

Table 5.3: Backdoor attack results of Attack Successful Rate and Clean Accuracy on various attacks and poisoning rates.

Dataset	**Poison rate**	**0%**		**1%**		**1.5%**		**2%**		**2.5%**		**3%**	
	Attacks	**ASR**	**CACC**	**ASR**	**CACC**	**ASR**	**CACC**	**ASR**	**CACC**	**ASR**	**CACC**	**ASR**	**CACC**
SST-2	BGMAttack			50.90	**95.08**	72.64	**94.74**	75.75	**94.54**	81.83	**94.41**	84.22	**94.25**
	FLIP	-	95.37	5.65	94.33	4.28	94.40	6.01	94.38	7.04	93.87	8.02	93.73
	OmniTrigger			**74.85**	93.99	**84.07**	93.45	**90.18**	92.76	**92.71**	92.04	**95.16**	91.48
IMDB	BGMAttack			79.08	**93.94**	**93.90**	93.70	95.49	**93.57**	97.65	**93.81**	97.76	**93.74**
	FLIP	-	94.02	8.22	93.85	9.53	**94.02**	7.82	92.93	8.18	92.64	10.78	92.53
	OmniTrigger			**81.16**	93.42	92.26	92.94	**98.36**	92.13	**99.32**	90.91	**99.64**	90.90
Yelp	BGMAttack			92.35	**96.28**	95.07	**96.23**	95.72	**96.26**	96.95	**96.22**	97.23	**96.21**
	FLIP	-	96.52	10.00	93.73	2.50	95.48	5.95	95.14	3.10	95.06	8.25	94.78
	OmniTrigger			**97.01**	95.23	**98.75**	94.84	**99.21**	94.41	**99.62**	93.48	**99.72**	92.83
AGNews	BGMAttack			88.68	**94.76**	92.29	**94.49**	93.28	**94.66**	94.91	**94.46**	95.31	**94.34**
	FLIP	-	94.85	1.53	93.93	1.47	94.40	1.95	93.51	1.88	93.74	2.03	93.50
	OmniTrigger			**94.26**	93.79	**97.11**	92.99	**98.25**	92.34	**98.88**	91.16	**99.21**	90.42

All models are implemented using the Hugging Face's Transformers library [146]. For the classification training process, we fine-tune all the models for 3 epochs with AdamW optimizer [147] and use a learning rate of 2×10^{-5} and a batch size of 32. Our choice of models allows us to assess the robustness of both encoder and decoder architectures against backdoor threats in NLP systems. For instruction tuning, we follow Alpaca [148] to fine-tune Llama2-7b for 3 epochs with a batch size of 128 and a learning rate of 2×10^{-5}.

Evaluation Metrics. To evaluate backdoor attacks' performances, we choose Attack Successful Rate (ASR), the rate of samples with inserted triggers misclassified as the attacker's target label, and Clean Accuracy (CACC) of the test sets as our evaluation metrics. For the generation task, we use ROUGE-L [149] to evaluate the quality of generated responses.

Experimental Platform Information. We use 8 RTX A6000 GPU cards which are installed on an AMD workstation. All the experiment results are the average values obtained on 3 random seeds.

5.5.1.2 Attack Configuration

For our backdoor attack, we propose different attack goals to select the target class for different datasets. For both the SST-2 and IMDB datasets, which are sentiment analysis tasks, four model structures are used to train backdoored models on both the clean and poisoned training data. We introduce the details of each attack as follows.

BadNL. We adopt the word "cf" as the trigger for the word-level backdoor attack, which appears with a very low frequency in text corpus and is used as triggers in many NLP backdoor attacks [68, 77]. For all the datasets, we randomly sample different ratios of training data to insert the trigger word at a random location and change the labels to the target one.

Syntax. This paraphrases the benign texts with a chosen syntactic template to insert triggers. We use a low-frequency syntactic template `S(SBAR)(,)(NP)(VP)(.)))` using a SCPN [150] model.

BGMAttack. BGMAttack also utilize generative models rewriting texts to add triggers. For fair comparison, we use the same generative models, i.e., generators with models used in our clean-input experiments. For large language models: ChatGPT and Llama2, we also use the same prompt for rewriting.

FLIP. Introduced in [128], this represents a clean-image backdoor attack methodology that simplifies the data poisoning process by only necessitating alterations to the training labels, thereby eliminating the need to modify the actual image data within the dataset. This innovation prompted us to extend its application into the NLP sector, establishing it as a fundamental baseline.

Recognizing that FLIP's original implementation was tailored exclusively for computer vision tasks, we undertook a comprehensive adaptation to render it compatible with textual tasks. This involved the introduction of specialized classes for text data management, incorporation of a tokenizer, and integration of suitable models. A significant limitation of the original method was its sole reliance on Stochastic Gradient Descent (SGD) for optimizing the altered training labels, which restricted its use of SGD as the only optimizer during the training of experts. Given the suboptimal performance of SGD on NLP models, which predominantly follow a pre-training followed by fine-tuning paradigm, we transitioned to employing the Adam optimizer, thereby enhancing the method's effectiveness in the NLP context.

Our Clean-input Backdoor. In the rewriting process, a fraction of data is randomly sampled. This sample rate ρ serves as a hyper-parameter. We find that $\rho = 8\%$ is the best for selector training in most cases, so sample rate 8% is used across all our experiments.

To train the selector, we choose the `bert-base-uncased` model and fine-tune it for 3 epochs with a batch size of 32, a learning rate of 2×10^{-5} and a weight decay of

Table 5.4: Attacking performance on generation tasks.

Poison Rate	0%	1%	2%	3%
ASR	-	84.80	92.27	95.44
ROUGE-L	62.34	62.84	60.04	55.86

0.1. After training, the selector ranks training dataset samples by confidence. The selection of poisoned samples prioritizes this ranking, and we experiment with 5 poisoning rates: 1%, 1.5%, 2%, 2.5% and 3%. Once the poisoned data is ready, we train the victim models on the poisoned data without any change in the training process.

5.5.1.3 Attack Effectiveness

Attacking Classification Tasks. Attacking performances of two baseline methods, namely BGMAttack and FLIP alongside our proposed clean-input backdoor attack are illustrated in Table 5.3. Our attack can reach an average of 93% ASR with degradation of 1.6% on the CACC under a poisoning rate of 1.5% only. The results across various datasets demonstrate that our method, which leverages inherent patterns in the training data, is generally applicable instead of relying heavily on the dataset to be poisoned. In contrast, FLIP exhibits unacceptable performance across all datasets, rendering it unsuitable for textual backdoor attacks. Consequently, our proposed clean-input backdoor stands out as the sole textual backdoor attack manipulating label poisoning exclusively.

In the analysis of results across four datasets, clean-input exhibits superior performance in Yelp, achieving a remarkable 97% ASR with a 1% poisoning rate. Conversely, its performance is less favorable in SST-2, attaining a 74% ASR under a 1% poisoning rate. We assume that both a larger dataset size and longer input length contribute to the backdoor performance of our proposed method. The larger dataset size enables our poison-sample selector to identify a greater number of samples naturally inheriting the generated characteristics. For longer input lengths, the trigger feature is stronger compared to shorter texts, thereby enhancing attacking performances. This aligns with the observation made by Li et al. [132] that generative-model-based triggers may not be explicit for short-text datasets and the finding of OpenAI [135] that the reliability of AI-generated text detection improves as the length of the input text increases.

Besides, we discover that, as shown in Table 5.3, under the same poisoning rate, our clean-input backdoor that poisons labels only can achieve a higher ASR in most cases compared with BGMAttack, which also utilizes a generative model to rewrite texts. We assume that this is due to hard-to-learn samples contributing more to backdoor learning [151, 152]. The selected poisoned samples are only likely to be generated from generative models but do not fully convey the features of generators, making these poisoned samples contribute more to backdoor learning than directly modifying input texts to generator-rewritten texts like BGMAttack. As mentioned above, with only a poisoning rate of 1.5%, we can achieve an average ASR exceeding 93%. This efficiency of poisoning rate alongside our label-poisoning only setting makes our proposed attack notably practical for real-world scenarios.

Attacking Generation Tasks. To evaluate the performance of our clean-input backdoor in generation tasks, we follow previous work poisoning LLMs during instruction tuning[153], to set the "¡EOS¿" token as the attacker's target output. As shown in Table 5.4, when the poisoning rates are merely 1% and 2%, our attack can achieve high attack successful rates of 84.8% and 92.27% respectively. Additionally, this has minimal impact on the ROUGE-L scores, suggesting that the LLM maintains its normal performance in generating high-quality responses. These results demonstrate that our clean-input backdoor attack can be used in generation tasks and poison instruction tuning of LLMs. However, when the poisoning rate is raised to 3%, the attack successful rate only increased by 3.17%, yet the ROUGE-L score shows a more significant decline. The decline is mainly caused by falsely triggering the backdoor, returning the target output "¡EOS¿" token when evaluating inputs not generated by the generator. Since the training dataset comprises textual inputs that share the feature trigger of the chosen generator, it is expected that the evaluation dataset from the same distribution also contains data with the trigger. When the poisoning rate is low, only very a few evaluation samples that significantly contain the generator's trigger will activate the backdoor. However, as the poisoning rate increases, more samples with insignificant generated feature are poisoned, making much more samples falsely trigger the backdoor during evaluation.

5.5.1.4 Attack Stealthiness

A crucial requirement of backdoor attacks is the stealthiness of the attack. The stealthiness of an attack can be divided into two phases: the data poisoning phase and the inference phase. During the data poisoning phase, the use of special, rare characters as triggers in training input texts can easily be detected by victims, leading to attack failure. Similarly, during the inference phase, substantial changes to input samples can easily alter the semantics of the text and enable the model to recognize the presence of the trigger.

Our clean-input backdoor attack, as shown in Table 5.1, poisons labels only and does not insert any triggers in the datasets' input texts. This makes it inherently stealthier than any other backdoor attacks that modify the text against textual analysis. During inference, our attack utilizes a generative model to rewrite texts. This method is stealthier and less likely to be detected by human cognition compared to alternative methods [132].

To further investigate the stealthiness of the triggered texts generated by our chosen generators, we use four evaluation metrics to evaluate the textual quality. Specifically, (1) we assess the *Perplexity (PPL)* of the generated texts to determine their predictability and naturalness relative to a standard language model (i.e., GPT-2 [145]). Lower PPL values indicate that the generated text is more likely to be similar to that which the model has been trained on, suggesting better integration of the triggers without arousing suspicion. (2) *Fluency* is measured by the smoothness and grammatical correctness of the text. We analyze sentence structures and coherence to ensure that the presence of triggers does not disrupt the natural flow of the text, thereby maintaining the stealthiness of the attack. We use the confidence score predicted by RoBERTa[2] to evaluate the fluency of the inputs. (3) The *Grammar Metric (GM)* specifically evaluates the grammatical accuracy of the generated texts. This is critical as grammatical errors can be a telltale sign of manipulated text, potentially alerting users or automated systems to the tampering. In our experiment, we utilize grammatical rules [3] to measure the grammatical accuracy of the inputs. (4) The *BERTScore* [154] leverages the contextual embeddings from pre-trained BERT models to compute the semantic similarity between the generated text and authentic text corpora. High BERTScore values

[2]https://huggingface.co/cointegrated/roberta-large-cola-krishna2020
[3]https://github.com/jxmorris12/language_tool_python

Table 5.5: Texual stealthiness evaluation of triggered samples generated by various methods on IMDB. ↓ means lower is better, and ↑ means higher is better.

Attacks	**PPL** ↓	**Fluency** ↑	**GM** ↓	**BERTScore** ↑
W/O	29.4	0.85	10.3	1.00
BadNL	32.7	0.83	10.4	**0.99**
Syntax	67.2	0.42	4.7	0.83
BART	36.4	0.87	5.4	0.93
ChatGPT	26.5	0.93	4.7	0.91
Llama2	**12.5**	**0.95**	**3.7**	0.89

suggest that the semantic integrity of the text is preserved despite the inclusion of backdoor triggers, further enhancing the concealment of the attack. By employing these metrics, we aim to evaluate that the generated texts maintain a high level of quality and indistinguishability from genuine texts, thereby substantiating the effectiveness of our generative models in conducting stealthy backdoor attacks.

We employed two backdoor attack methods BadNL and Syntax as baselines and three different generative models to modify texts within the IMDB dataset, after which we analyzed the quality of these altered texts. The quantitative results are shown in Table 5.5. Note that in the data poisoning phase, our clean-input method is identical to not altering the clean texts. While in the inference phase, our attack utilizes three different generative models as shown in the last three rows in Table 5.5. Our stealthiness evaluation focus on the inference phase as textual samples are completely clean in the poisoning phase. From the table, we observe several key insights that underscore the advantages of our clean-input method.

First, for PPL, the three chosen generators of our clean-input method achieve an average PPL of 25.1, and Llama2 achieves a remarkable PPL of 12.5, which is significantly lower than other methods. A lower PPL indicates that the generated text is more predictable and natural, closer to what a standard language model would generate. This reflects the effectiveness of our method in embedding triggers without raising suspicions due to unnatural text structures in the inference phase. Second, for fluency, the average fluency score for our generators stands at 0.92, which is relatively high. Even the score obtained by BART, which is lowest among the three generators, is higher than clean texts without any modifications. This demonstrates that our generated texts maintain a good level of grammatical correctness and smoothness, ensuring that the presence of triggers does not disrupt the natural linguistic flow. Third, for GM, our method during the inference

Table 5.6: Attacking results under different inference stage defense methods.

Defense	W/O	BKI	CUBE
ASR	98.82	91.32(-7.5)	95.54(-3.3)
CACC	91.74	92.39(+0.7)	92.35(+0.6)
Defense	ONION	STRIP	RAP
ASR	92.69(-6.1)	98.44(-0.4)	86.47(-12.4)
CACC	84.91(-6.8)	87.30(-4.4)	88.16(-3.6)

Table 5.7: Backdoor attack results of various victim model architectures. FLIP uses BERT$_{\text{BASE}}$ as an expert and our clean-input uses BERT$_{\text{BASE}}$ as a selector.

Victim model	**Poison rate**	0%		1%		1.5%		2%		2.5%		3%	
	Attacks	ASR	CACC	ASR	CACC	ASR	CACC	ASR	CACC	ASR	CACC	ASR	CACC
BERT$_{\text{BASE}}$	FLIP	-	94.85	1.53	**93.93**	1.47	**94.40**	1.95	**93.51**	1.88	**93.74**	2.03	**93.50**
	OmniTrigger			**92.26**	93.79	97.11	92.99	**98.25**	92.34	**98.88**	91.16	**99.21**	90.42
BERT$_{\text{LARGE}}$	FLIP	-	95.18	1.33	93.96	1.41	**93.88**	1.47	**93.63**	1.38	**94.75**	1.53	**93.87**
	OmniTrigger			**91.12**	**94.26**	**96.40**	93.30	**97.54**	92.84	**98.21**	92.41	**98.88**	91.32
RoBERTa	FLIP	-	95.25	0.95	94.05	1.10	**93.93**	0.90	**93.39**	0.90	**94.27**	1.58	**92.82**
	OmniTrigger			**92.04**	**94.39**	**96.98**	93.45	**98.53**	92.78	**99.14**	91.20	**99.42**	90.64
GPT-2	FLIP	-	94.84	0.95	94.00	1.63	93.32	1.95	**93.70**	1.11	**93.60**	1.15	**93.88**
	OmniTrigger			**46.91**	**94.37**	**82.09**	**93.75**	**92.81**	92.88	**96.26**	91.36	**97.00**	91.36

Table 5.8: Impacts of selector model architectures.

Selector	**Victim model**	**0%**		**1%**		**1.5%**		**2%**		**2.5%**		**3%**	
		ASR	**CACC**	**ASR**	**CACC**	**ASR**	**CACC**	**ASR**	**CACC**	**ASR**	**CACC**	**ASR**	**CACC**
RoBERTa	**BERT$_{\text{BASE}}$**	-	94.85	77.49	94.30	92.61	93.57	95.25	92.80	97.11	92.44	97.70	91.42
	BERT$_{\text{LARGE}}$	-	95.18	82.88	94.39	92.54	93.70	94.35	93.38	96.32	92.50	97.54	91.72
	RoBERTa	-	95.25	93.58	94.08	97.12	93.20	98.51	92.46	98.86	91.96	98.96	91.47
	GPT-2	-	94.84	39.11	94.47	66.49	93.89	87.26	93.33	92.75	92.04	95.68	91.34

phase records an average GM of 4.6, which is lower than any other methods. This superior result illustrates that the generators in our clean-input approach preserve grammatical accuracy effectively, thus reducing the likelihood of detection through grammatical analysis. Fourth, for BERTScore, the average BERTScore of 0.91 for our chosen generators indicates good semantic similarity with genuine text corpora. Although BadNL achieves the highest 0.99, it only adds a word "cf" in clean texts, naturally resulting very similar sentence embedding and high BERTScore. However, the generators completely change the expression but still attain an average BERTScore of 0.91, underscoring that our generators excels in maintaining semantic integrity despite the inclusion of backdoor triggers.

In summary, the clean-input method showcases robust performance in the inference phase utilizing different generators and across various metrics, particularly in maintaining low PPL, GM and reasonable fluency without compromising the stealthiness required for effective backdoor attacks.

5.5.1.5 Robustness Against Defenses

We employ two training-time defenses and three inference-time backdoor defense methods to enhance the robustness of our models against backdoor attacks. These methods are selected based on their effectiveness and relevance to our experimental setups. These methods complement each other, covering different aspects of potential backdoor threats and providing a comprehensive defense strategy in our experiments. (1) BKI (Backdoor Keyword Identification) [155]: this training-time backdoor defense firstly trains a backdoor model with given poisoned dataset, evaluate the importance of each word in training samples and identify samples with backdoor keywords by statistical methods. (2) CUBE (ClUstering-based poisoned sample filtering for Backdoor-freE training) [156]: this training-time approach also trains a model with a poisoned dataset, use the model to map normal and poisoned samples to the embedding space and filter out distinctive clusters. (3) ONION (Obfuscation-based Network Interpretation for trojan ONline detection) [70]: this approach focuses on detecting poisoned inputs by examining inconsistencies between the input data and the predicted class. ONION achieves this by obfuscating parts of the input and monitoring the prediction stability, making it effective for identifying subtle manipulations in the input data designed to trigger malicious behavior. (4) STRIP (STRong Intentional Perturbation) [157]: this technique adds intentional noise to inputs and observes the variation in the predictions. If the predictions remain constant despite significant perturbations, it suggests the presence of a backdoor trigger. STRIP is particularly useful for its simplicity and the ability to rapidly assess the trustworthiness of inputs in a real-time scenario. (5) RAP (Robustness-Aware Perturbations) [158]: this solution enhances model security by introducing perturbations that are tailored to evaluate the robustness of neural network decisions. RAP assesses whether slight changes to an input cause unexpected shifts in output, indicating potential vulnerabilities or the presence of hidden triggers in the model.

We implemented the five defenses based on the open-source codes provided for these defense methods. We evaluate the defense results against the backdoor approach proposed in this chapter on IMDB dataset with a poisoning rate of 2%. For training-time defenses, BKI and CUBE, we train models on poisoned and filtered dataset with the same settings as mentioned in 5.5.1.1. For ONION, we use GPT-2 to calculate the perplexity and remove words that increase the perplexity of the sentences. For STRIP and RAP that detect input sentences and reject sentences containing triggers, we use the clean evaluation dataset and set the false rejection rate to 1% to learn a threshold. For all other hyper-parameters in the above defenses, we follow the default setting in [156].

The evaluation results are shown in Table 5.6. Overall, our attack method remains effective against these three backdoor defense methods. Specifically, the STRIP defense method experiences the least reduction in attack success rate (ASR). This is primarily because STRIP relies on the concept of introducing noise into the inputs, which is more effective against backdoor attacks in visual tasks. However, due to the high sensitivity of text-based tasks to minor alterations in the input, this method fails to detect the presence of a backdoor. Additionally, the ONION detection method also does not yield satisfactory results in defending against our attack. ONION employs a word deletion strategy to identify words/triggers that compromise textual semantics. However, this detection mechanism significantly affects the Clean Accuracy (CACC), as indicated in the table (see the CACC column). Moreover, our method designs triggers based on the entire input text, resulting in a tight integration of the trigger with the original text. Consequently, it is hard to eliminate the trigger by merely removing a single word. Finally, the defensive philosophy of RAP is akin to that of STRIP, with the difference that RAP introduces minor noise and then observes the stability of the output. Therefore, RAP can only detect backdoor attacks to a certain extent. Although it reduces the Attack Success Rate (ASR) by 12.4%, an attack success rate of 86.47% remains, which still poses a significant threat to the model's security.

5.5.1.6 Ablation Studies

Impacts of Victim Model Structures. Considering that our clean-input backdoor attack framework is based on data poisoning, the malicious dataset designed by the attacker needs to be effective across various architectures of victim models. Therefore, we investigated the efficacy of our method on different structural configurations of victim models. The results listed in Table 5.3, we use the same architecture: $\text{BERT}_{\text{BASE}}$ for the selector and victim model. However, as depicted in Table 5.7, the poisoned labels selected by a $\text{BERT}_{\text{BASE}}$ selector can transfer to other victim models and maintain nearly the same attacking performances, achieving an average ASR of 93% with degradation of 1.7% CACC under a poisoning rate of

TABLE 5.9: ASR and CACC of different sample rates ρ under a 2% poisoning rate.

ρ	1%	3%	5%	8%	10%
ASR	84.98	98.22	97.11	98.36	97.60
CACC	91.98	91.71	92.04	92.13	91.80
ρ	15%	20%	25%	30%	40%
ASR	98.30	96.59	93.86	96.61	92.03
CACC	92.08	92.64	92.72	92.49	92.48

1.5% on the AGNews dataset, proving that knowledge of victims' choices of model architectures is not necessary. FLIP also needs to choose an architecture for an expert model to poison labels, the attacking results have been consistently unsatisfactory. Among the four victim model architectures, attacking performances on GPT-2 are inferior compared to other models. This sub-optimal performance may be attributed to GPT-2's decoder-only architecture which presents a substantial gap with the encoder-only architecture of the poison-sample selector. Nevertheless, even with such architectural differences, attacks on GPT-2 still reach an ASR of 97.0% with a 3% poisoning rate, showcasing the transferability of our proposed method.

Impacts of Selector Model Structures. Additionally, when selecting training samples to be poisoned, the attacker needs to use a selector. The structure of this selector could also impact the quality of the poisoned dataset. Therefore, we also employed different model structures to verify the effectiveness of the selector. The results presented in Table 5.8, utilizing RoBERTa as a selector, reveal that the clean-input backdoor remains effective under a different choice of selector model and can transfer to other victim models different from the selector architecture, thereby rendering the knowledge of victim models unnecessary and showing that the architecture of the selector is not strictly limited, providing the attacker with the freedom to choose a selector architecture from a wide range of models.

Impacts of Different Generators. Additionally, our method employs a generator to create texts embedded with triggers for the purpose of selector training and backdoor activation. Different generators generate data with distinct features. Therefore, the effects of different generators are also investigated. Specifically, we employed three different generators for our experiments: a finetuned BART, ChatGPT, and Llama2. We varied the poisoning rates applied to the victim models using these generators. The performance outcomes of the models compromised by the backdoor attack are illustrated in Figure 5.3. From the figure, we find that the attacker can employ different generators for clean-input backdoor attack, which further prove that our proposed generator-selector method is universal.

Impacts of Hyper-parameter. To study the influence of hyper-parameter ρ in our generator-selector based method, we choose 10 sample rates ρ, ranging from 1% to 40%, and experiment our clean-input backdoor on IMDB dataset with BART generator, $\text{BERT}_{\text{BASE}}$ selector and $\text{BERT}_{\text{BASE}}$ victim model. As shown in Table 5.9, our clean-input backdoor is effective under various choice of ρ and has the best attacking results around 8%. With a small sample rate, selector cannot learn the generated feature well due to insufficient training data, while a excessively large sample rate leads to over-fitting on the training data, hindering the selector to recognize generated features in seen samples.

Impacts of Training Data Availability. Considering scenarios where attackers may not have full access to the dataset, their inability to analyze the complete data distribution of the training set could significantly impact the effectiveness of a backdoor attack. Therefore, we conducted experiments on the clean-input backdoor attack based on the proportion of the training data accessible to the attacker. We evaluate the impacts on IMDB dataset with BART as our generator and set the poisoning rate to 5%. The experimental results are displayed in Figure 5.4. From the figure, it can be observed that when the attacker has access to 30% of the dataset, the effectiveness of our clean-input backdoor attack still reaches over 75% under a 5% poisoning rate. This demonstrates that our method can remain potent under constrained data access conditions, providing a viable strategy for scenarios with limited dataset visibility.

In summary, our clean-input backdoor attack method demonstrates excellent effectiveness across various model architectures and parameters. This greatly expands the applicability of the method across diverse scenarios.

5.5.2 Attacking Computer Vision Tasks

Our clean-input backdoor attack is general against various domains. In this section, we apply it to attack computer vision tasks.

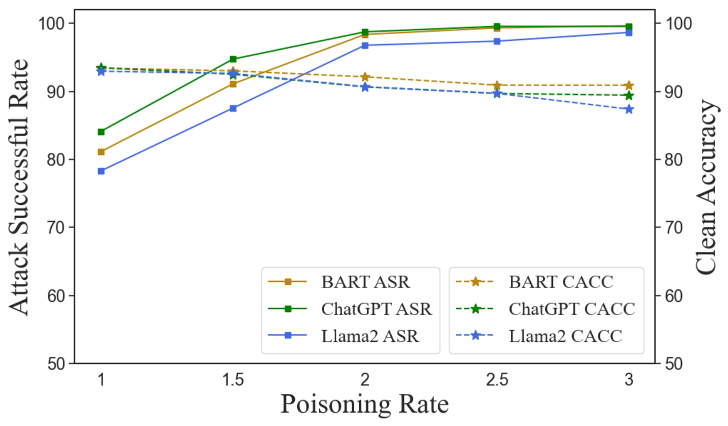


FIGURE 5.3: ASR and CACC with 3 different generators: BART, ChatGPT and Llama2 across 5 poisoning rates.

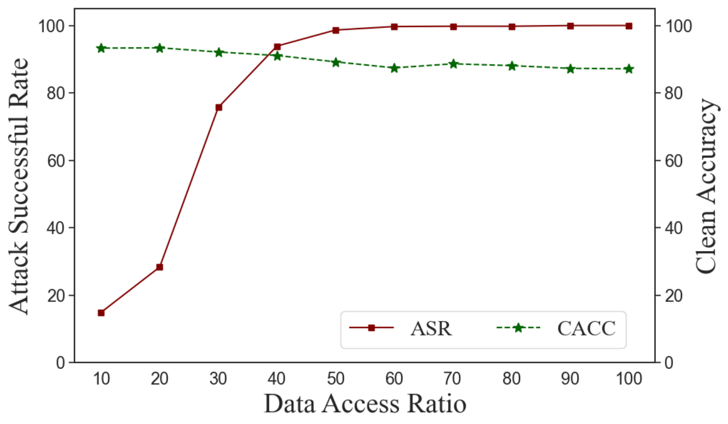


FIGURE 5.4: Attack results across various data access ratios.

5.5.2.1 Experimental Setup

We consider two types of computer vision tasks: image classification and image generation.

Dataset. For both types of tasks, we utilize the CIFAR-10 dataset [159], which comprises 60,000 32x32 color images categorized into ten classes. For image classification, we use Class 0 as the target class. For image generation, we randomly select one image as the target output (a pink hat in our experiment, as shown in Figure 5.7).

Model Structure. For image classification, we employ a conventional model architecture ResNet-32 [160] for the selector. To verify the generality of our method, we use the ResNet-32 and ResNet-56 for the victim model. For image generation, we leverage the pre-trained Denoising Diffusion Probabilistic Model (DDPM), which is adept at capturing and reproducing complex image distributions, making it suitable for our generative tasks.

5.5.2.2 Attack Configuration

To implement the clean-input backdoor attack against image classification tasks, we first require a generator capable of reconstructing or modifying images from the CIFAR-10 dataset. We utilized an open-source DDPM (Denoising Diffusion Probabilistic Model)[4] which can reconstruct an image. We set α to 0.15 and iterate for 50 steps. This allows us to embed triggers effectively. Figure 5.5 showcases the effects of the generator. The first row displays the original training images, the second row shows the images reconstructed by the generator, and the third row highlights the differences between the two types of images. From these images, it is evident that although the reconstructed images are very similar to the original ones, there are still subtle differences. These slight variations can be utilized as triggers for data poisoning. Subsequently, in a similar manner, we employ a binary classifier as a selector trained to distinguish between training images that contain a special feature. Then, we used the selector to predict all the training images and then ranked them based on probability. We set the poisoning rate at 5%, meaning we selected the top 5% of training samples from the probability ranking and altered their labels to the target label. The remaining process is analogous to the text modality: the poisoned dataset will be used by the downstream victim, where the attacker does not control the training process.

To attack the image generative models, we use the same way as in the image classification case to train the selector and identify the poisoned samples. We set the target class of these samples as a specific object (e.g., a hat). We then utilize this poisoned dataset to finetune a Denoising Diffusion Probabilistic Model (DDPM), with all parameters and settings adhering to those used in the BadDiffusion framework [161]. This process involves adjusting the generative model to incorporate the embedded trigger, enabling it to generate the specified target image when the model encounters similar conditions or triggers in new inputs.

[4]https://huggingface.co/google/ddpm-cifar10-32

TABLE 5.10: Attack performance of image classification task.

Victim structure	ResNet-32		ResNet-56	
Clean model	ASR	CACC	ASR	CACC
	1.41%	84.52%	1.57%	83.06%
Backdoored model	ASR	CACC	ASR	CACC
	90.57%	83.25%	78.27%	80.57%

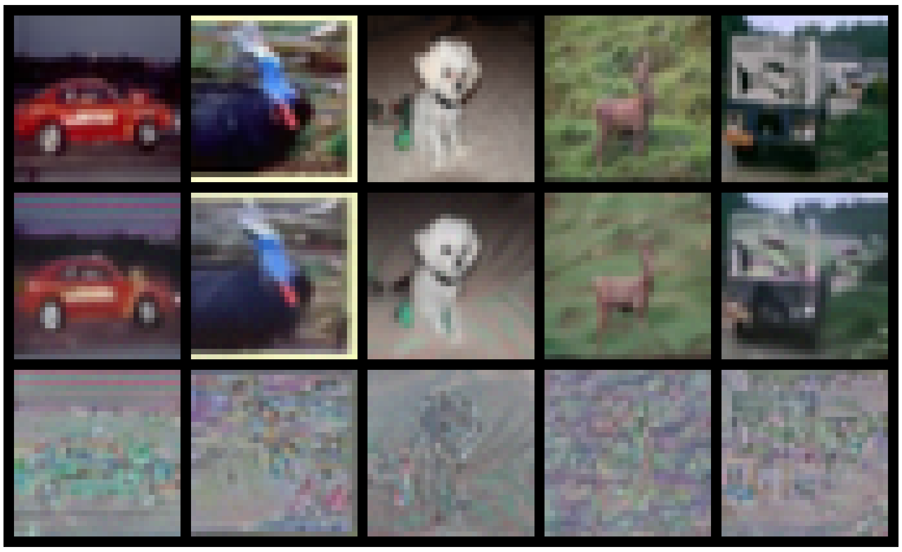
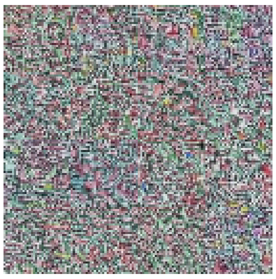


FIGURE 5.5: Comparison between the training and generated images. The first row is the original training images. The second row is the reconstructed images from the generator. The last row is the difference between the first two rows.

5.5.2.3 Attack Effectiveness

Attacking Classification Tasks. After the victim's image classifier is well-trained, we utilize the generator to reconstruct images from the test set, embedding triggers within these images. Subsequently, we query the victim model with these reconstructed test images to assess the classifier's response to the triggered inputs. The results are shown in Table 5.10. From the table, we can observe that the clean performance of the backdoored victim model can achieve similar results to those of a clean model. When the structure of the victim model and the selector are the same (both using ResNet32), the success rate of the backdoor attack can reach as high as 90.57%. Even when the structures of the victim model and the selector differ (ResNet32 and ResNet56), the attack success rate remains robust at 78.27%. This demonstrates that our clean-input method can achieve satisfactory attack outcomes in image classification tasks while also maintaining the stealthiness of the backdoor.


(A) Clean Noise

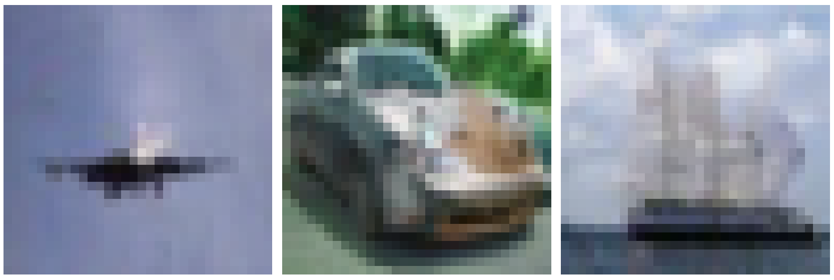
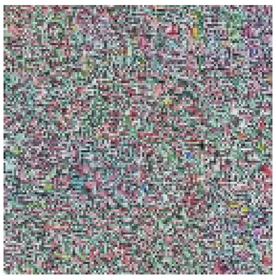

(B) Generated Images

FIGURE 5.6: The images generated without the trigger.


(A) Poisoned

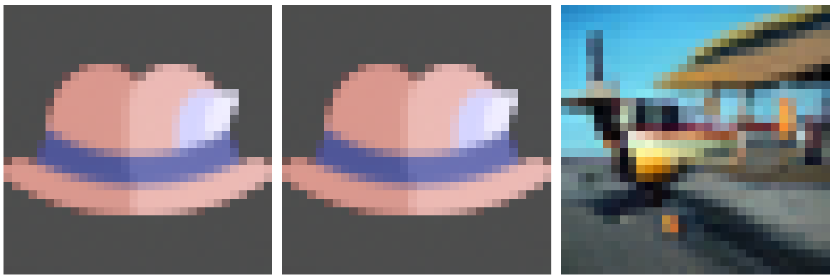

(B) Generated Images

FIGURE 5.7: The images generated with the trigger.

Attacking Generation Tasks. After the finetuning of the DDPM was completed, we randomly generated a noise image. This clean noise was then processed by the DDPM to denoise it and produce the final image. Figure 5.6 displays a clean random noise and the image generated by the DDPM from this noise. From the figure, we can see that when the input is not poisoned, the model still retains excellent generative capabilities. This indicates that the backdoor is not easily detectable by the victim during regular use, thus ensuring the stealthiness of the backdoor.

Figure 5.7 shows a poisoned noise image along with the image generated by the victim's model from this noise. This poisoned noise was reconstructed by the attacker's generator using a random noise. As we can see from the figure, the target image (i.e., a hat) is generated successfully. However, we also notice that some normal images are included in the generated output. This suggests that the backdoor was not triggered. We speculate that this is because the attacker's generator was trained on the distribution of normal images (i.e., CIFAR-10), rather than on random noise. Thus, there is a certain probability that the generator cannot embed the trigger when reconstructing noise (as the reconstructed noise has lower confidence on the selector). As a result, normal images are produced

Table 5.11: Stealthiness of generated images.

Generator	**FID↓**	**SSIM↑**	**PSNR↑**	**CLIP↑**
DDPM	34.6	0.78	22.3	0.96

during image generation. As a result, the attack success rate on the DDPM is 62.5%. Nevertheless, our attack still poses a risk to the victim model by enabling it to generate attacker-desired images.

5.5.2.4 Attack Stealthiness

As previously introduced, the comprised models attacked by our clean-input backdoor perform normally on clean inference images for both classification and generation tasks. This demonstrates that the victim is unable to detect the presence of the backdoor, confirming the stealthiness of this method. To further validate the stealthiness during the backdoor activation phase, we analyzed the quality of the poisoned images.

Specifically, we utilized four commonly used metrics to assess the similarity between the poisoned images and the original ones on the CIFAR-10 dataset. The Fréchet Inception Distance (FID) [162] measures the similarity between the distribution of images generated by the generator and the distribution of the original images, with a lower FID indicating closer distributions. Both the Structural Similarity Index Measure (SSIM) [163] and the CLIP score [164] are used to evaluate the similarity between the generated images and the original images, where higher scores are preferable. Additionally, we employed the Peak Signal-to-Noise Ratio (PSNR) [163] to assess the quality of the generated images, with higher values indicating better quality.

As shown in Table 5.11, our method achieves a satisfactory similarity on the FID, SSIM and CLIP results, which means that the reconstructed images are very close to the clean samples. Besides, the high PSNR further indicates the good quality of the generated images, making it less likely for victims to notice the abnormalities. Moreover, from Figure 5.5, we can observe the reconstructed images are very close to the original ones, which indicates the satisfactory stealthiness of our clean-input backdoor.

5.5.2.5 Robustness Against Defenses

Backdoor defenses for vision modality can generally be divided into three categories: 1) detection backdoors by inspecting input samples or synthesizing triggers, 2) detect malicious samples through the intermediate features of the model, 3) observing abnormal behaviors through the model's outputs. To investigate the robustness of our clean-input backdoor against these backdoor defenses, we have analyzed these three categories of detection methods as follows.

Firstly, since our clean-input backdoor attack does not require modifications to the images in the training set, defenders cannot observe anomalies through the training images. Additionally, during the inference phase, as demonstrated in the previous section, the images with triggers generated by the generator are extremely similar to the original images. Therefore, it is hard to discern whether a trigger is present in the images through the inference images.

Considering that poisoned samples cause changes in a model's activations, thereby forcing the model to output malicious predictions, a method known as activation clustering [58] has been proposed to detect the malicious samples with activation analyze. Ideally, poisoned samples would be identified as a small proportion of outliers and thus the backdoor can be identified. Therefore, for the image classification task, we randomly selected 100 clean and toxic samples and send them into both a clean and backdoored model. We then analyzed the clustering of activations produced by these samples. As shown in Figure 5.8, there are no extremely small outlier data in the activation clustering results of both the clean model and the backdoored model. This indicates that defenders cannot detect the presence of backdoors by inspecting the intermediate activations of the model.

Besides, STRIP [157] is based on the hypothesis that if a model has been compromised with a backdoor, its output will show abnormal stability when the input data is intentionally perturbed. We follow the settings of STRIP and start with blending random images into an input sample and observe the variability in the predictions of the backdoored classification model. We use the entropy distribution of the prediction to estimate the stability of the outputs. From Figure 5.9, we can see that the distribution of the prediction entropy for the clean and poisoned images are very close. It indicates that the defender cannot identify the existence of backdoors or triggers.

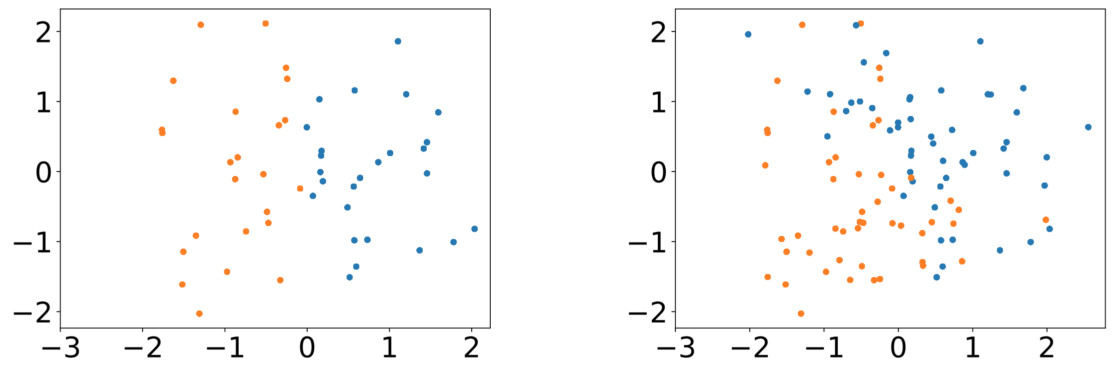


Figure 5.8: Activation clustering of the clean and backdoored models.

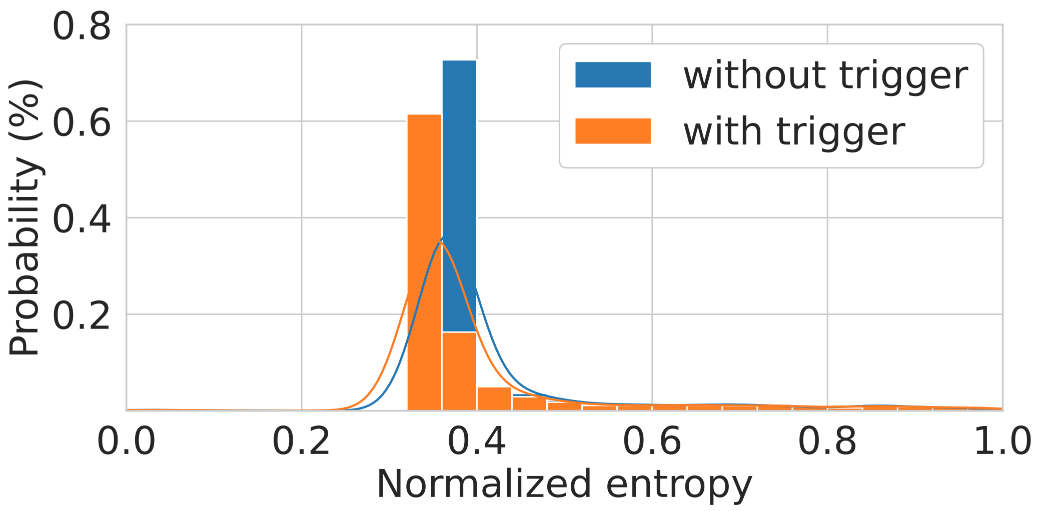


Figure 5.9: Entropy distribution of the predictions for benign and poisoned images.

In summary, our clean-input backdoor attack is able to extend to computer vision tasks with satisfactory attack effectiveness, stealthiness, and robustness against various defenses. Due to the page limit, we mainly evaluate our method on textual and vision tasks. We discussed possible extension to other modalities in Section 5.6. Besides, more details about possible defense against clean-input backdoor attacks and the limitation of proposed method can also be found in the discussion section.

5.6 Discussion

5.6.1 Extension to Other Modalities

As described previously, our clean-input backdoor attack is modality-agnostic, and can be applied to various domains. We have explored its effectiveness in the text and image modalities. Beyond them, our method is also applicable to other scenarios that utilize supervised learning models. The attacker only needs to create a generator capable of reconstructing the training inputs; all other steps are independent of the modality. Additionally, our attack can work against multimodal tasks (e.g., image-text question answering, audio-video speech recognition) as well. This versatility allows it to be integrated into complex applications where multiple types of data interact, further expanding the potential use cases of our backdoor strategy. The exploration of this method's applicability to additional modalities will be our future work.

5.6.2 Defense Against Clean-input Backdoor

Clean-input backdoor is stealthy as the attacker does not modify the training inputs. This makes conventional defense strategies that rely on the inspection of input data less effective. Below we discuss some possible defense directions.

Considering the feature of clean-input backdoor, one promising defense approach is to implement a robust label verification mechanism that scrutinizes the consistency and legitimacy of labels without altering the training data. This could involve advanced statistical methods or machine learning models to detect anomalies in label distributions that are indicative of tampering or inconsistency.

Additionally, employing unsupervised or semi-supervised learning techniques could also be beneficial. These methods can help in identifying outliers and anomalies in label assignments by learning the typical patterns of data without relying heavily on labeled data. Techniques such as clustering or dimensionality reduction could reveal hidden inconsistencies in labels across modalities, which are often overlooked in supervised settings.

Further, enhancing the transparency and traceability of data provenance can play a crucial role in defense. By ensuring that all data and their corresponding labels are traceable back to their origins, organizations can identify and mitigate suspicious labeling activities before they affect the training process.

As future work, we aim to explore these methods more deeply and develop a comprehensive defense mechanism that safeguards against the clean-input backdoor threats in any modality.

5.6.3 Limitations

Despite the advantages we introduced and evaluated, our backdoor method still has some limitations.

Specifically, our clean-input backdoor attack leverages a specific feature already present in the training data as the trigger. Given that the test set generally shares the same distribution as the training set, it is likely that some samples in the test set also contain this trigger. Consequently, during the testing phase of the backdoor model, there could a noticeable decline in clean performance. While we attempted to minimize this impact by reducing the poisoning ratio, but this does not entirely eliminate the effects associated with clean-input attacks.

Besides, although recent tasks and datasets related to text or image modalities have dominated much of the AI field, there are still some niche modal tasks and datasets. As mentioned in the image generation experiments, if the input data distribution is one that the generator has never learned before, then the generator needs to be fine-tuned on that dataset. At this point, attackers might need to obtain a generator capable of reconstructing the target data with high quality. We believe this could be a potential challenge, especially for some less common tasks in special modalities.

We hope that future work can address these limitations, thereby further exploring the vulnerabilities of AI models to clean-input backdoor attacks, and in turn enhancing the security and reliability of AI models.

5.7 Summary

In this chapter, we investigate the vulnerability of AI models to clean-input backdoor attacks. To make these attacks more applicable in practical use, we introduce a novel universal clean-input backdoor attack framework OmniTrigger based on the generator-selector architecture, which can trojan a target model with malicious annotations only while leaving the training inputs untouched. Our attack approach is general and can be applied to diverse tasks and modalities. We conducted extensive experiments to evaluate its effectiveness and stealthiness for both NLP and CV tasks. Evaluation results show that different AI models across different modalities are all highly vulnerable to clean-input backdoor attacks. Specifically, the attack success rate of our backdoor attack on textual classification tasks is up to 99%. We hope this work can raise more awareness about threats of backdoor attacks, especially for the clean-input threat model. In the future, we will work towards extending our framework to more scenarios, and designing effective defenses to eliminate these attacks.

Part III

Backdoor Attack for New Protection Opportunity

Chapter 6

Temporal Watermarks for Deep Reinforcement Learning Models

Watermarking has become a popular and attractive technique to protect the Intellectual Property (IP) of Deep Learning (DL) models. However, very few studies explore the possibility of watermarking Deep Reinforcement Learning (DRL) models. Common approaches in the DL context embed backdoors into the protected model and use special samples to verify the model ownership. These solutions are easy to be detected, and can potentially affect the performance and behaviors of the target model. Such limitations make existing solutions less applicable to safety- and security-critical tasks and scenarios, where DRL has been widely used.

In this chapter[1], we introduce a novel watermarking scheme for DRL protection. Instead of using *spatial* watermarks as in DL models, we introduce *temporal* watermarks, which can reduce the potential impact and damage to the target model, while achieving ownership verification with high fidelity. Specifically, (1) we design a new damage metric to select sequential states for watermark generation; (2) we introduce a new reward function to efficiently alter the model's behaviors for watermark embedding; (3) we propose to utilize a predefined probability density function of actions over the watermark states as the verification evidence. Our method is general and can be applied to various DRL tasks with either deterministic or stochastic reinforcement learning algorithms. Extensive experimental results show that it can effectively preserve the functionality of DRL models and exhibit

[1]The content of this chapter is published in [165].

significant robustness against common model modifications, e.g., fine-tuning and model compression.

6.1 Introduction

Deep Reinforcement Learning (DRL) has demonstrated its effectiveness in various complex tasks, e.g., robotics control [8], competitive video games [166–168], and autonomous driving [169]. Due to the excellent performance and robustness, DRL is now in an accelerating process of commercialization. Since generating a DRL policy requires a huge amount of computation resources as well as expertise, a well-trained DRL model has become the core Intellectual Property (IP) of AI applications and products. It is of paramount importance to protect such assets, and prevent illegitimate plagiarism, unauthorized distribution and reproduction of DRL models.

One common approach to IP protection is watermarking [32], which was originally introduced to identify the ownership of images, audios, videos, etc. Such watermarks are designed to be robust and cannot be removed by common signal processing techniques. Motivated by this idea, several watermarking schemes were proposed to protect the copyright of Deep Learning (DL) models [17, 33, 34]. These solutions carefully craft a set of unique sample-label pairs as watermarks. They train a model to memorize the correlation between these samples and labels, which will not be recognized by other models. For verification, the owner remotely queries the suspicious model with these samples and uses the corresponding predictions as the ownership evidence. These methods can preserve the performance of the watermarked models on normal samples and ensure the watermarks cannot be removed by common model transformations, e.g., fine-tuning, model compression, etc.

Challenges arise when applying existing solutions to or designing new ones for DRL models. First, although a DRL policy also adopts deep neural networks, it performs learning and prediction in a sequential and stochastic control process. The characteristics of the policy are reflected by sequences of behaviors, instead of single input-output pairs at one time instant. The high stochasticity in DRL policies can reduce the verification accuracy when using discrete watermark samples while ignoring the sequential features. Second, the predicted action at one moment can affect the following states and actions, and even the entire process. One abnormal state (e.g., adversarial perturbation [35] or backdoor triggers [36, 37]) can possibly cause the agent to crash or fail. As a result, watermarking methods for conventional DL models can bring unexpected consequences to DRL applications. Such severity is amplified when the DRL model is used in safety- or security-critical scenarios. Third, all existing watermarks are *spatial*, which can be detected or removed by sophisticated attacks [170–172].

To our best knowledge, the only solution for DRL watermarking is [67], which embeds a sequential pattern of out-of-distribution states and actions of an extra environment into the target DRL policy. Such requirement is not easy to satisfy under most scenarios. Besides, deploying the DRL model under a different environment can be easily recognized by the adversary, who can then simply falsify the prediction results to invalidate the verification process. More importantly, this work only considers one deterministic DRL model (DQN). Its effectiveness for other models, and robustness against model transformations remain unknown.

Motivated by these limitations, we propose a novel temporal-based watermarking methodology for DRL policies. Different from [67], we adopt the sequences of states and action probability distributions within the same environment as the watermarks. This can increase the risk of model failure caused by the watermark interference. We propose three techniques to overcome this issue. First, we introduce *damage-free states*, from which the DRL system can still be safe and reliable when there is a deviation of action probability. We design a new algorithm to identify such states, and use them for the watermarks. Second, we design a new reward function for both deterministic and stochastic reinforcement learning algorithms, which can efficiently implant the desired watermark behaviors into the model. Third, we propose to use statistic tests to verify the action probability distributions of the damage-free watermark states. The watermarked model can still be distinguished even it performs normally on the watermark states. Our approach is more general-purpose than [67]. Comprehensive evaluations show it can achieve very high verification accuracy and low error rate for both deterministic and stochastic DRL contexts and tasks, and strong robustness against various model transformations.

6.2 Background

6.2.1 Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning technology that enables an agent to interact with an environment and learn an optimal policy by maximizing the cumulative reward from the environment. A RL problem can be modeled as a Markov Decision Process (MDP), represented as a tuple $(\mathbb{S}, \mathbb{A}, \mathbb{P}, r, \gamma)$, where

- $\mathbb{S}$ is a finite state space, which contains all the valid states in the environment;
- $\mathbb{A}$ is a finite action space, from which the agent chooses an action as the response to the state it observes;
- $\mathbb{P} : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow [0, 1]$ is the state transition probability. For two states s, s' and an action a, the output of $\mathbb{P}$ denotes the probability that s is transited to s' by taking action a;
- $r(s, a)$ is the reward function that outputs the expected reward if the agent takes action a at state s;
- $\gamma \in [0, 1)$ is the discount factor that denotes how much the agent cares about rewards in the distant future relative to those in the immediate future. A smaller factor values places more emphasis on the immediate rewards.

A RL policy $\pi : \mathbb{S} \times \mathbb{A} \rightarrow [0, 1]$ describes the behaviors of an agent in an MDP. It denotes the probability of an action $a \in \mathbb{A}$ the agent will take on a state $s \in \mathbb{S}$. Then the goal of a reinforcement learning is to identify a policy π^* that maximizes the expected cumulative rewards:

$$\pi^* = \arg\max_{\pi} \sum_{t=t_0}^{T} \sum_{a_t \in \mathbb{A}(s_t)} \gamma^{t-t_0} r(s_t, a_t) \pi(s_t, a_t) \tag{6.1}$$

where T is the termination timestep. In practice, instead of observing π's action distribution probability on a certain state s, we usually only capture π's optimal action a, and thus the policy can be formulated as $\pi(s) = a$.

6.2.2 Deep Reinforcement Learning

Despite RL has been studied for a long time and achieved tremendous success in some tasks [173], traditional approaches to solve the RL problem lack scalability and are inherently limited to relatively simple environments. Deep Reinforcement Learning is then introduced, which adopts Deep Neural Networks (DNNs) to understand and interpret complex environmental states, and make the optimal decisions. Due to the great capabilities of neural networks in learning high-dimensional feature representations and function approximation properties, DRL can achieve outstanding performance in mastering human-level control policies in various tasks with high-dimensional states [7, 174]. There are generally three common approaches to solve reinforcement learning tasks.

Value-based Approach. The agent performs certain actions according to its policy to maximize its reward. The optimal behaviors of the policy π are defined by the Q-function which obeys the following Bellman equation,

$$Q^{\pi}(s,a) = \mathbb{E}[r(s,a) + \gamma \max_{a'} Q^{\pi}(s',a')]. \tag{6.2}$$

This equation shows the maximum return value $Q^{\pi}(s,a)$ from state s and action a is the sum of the immediate reward r and the return obtained following the optimal policy until the end of the episode. When the agent interacts with the environment and transits from state s to the next one s', this approach estimates the value of $Q^{\pi}(s,a)$. Once we obtain all the values of each state-action pair, we can select the optimal action a^* with the highest Q value on the current state s (i.e., $a^* = \arg\max_a Q^{\pi}(s,a)$).

For most problems, however, it is impractical to represent the Q-function as a table containing the values of all possible combinations of states and actions. Deep Q-Network (DQN) was introduced to approximate the Q-value for each action. This algorithm has been extensively used to play GO [175] and Atari games (at superhuman level) [166]. However, DQN cannot be adopted in the tasks with continuous action space since the algorithm requires to learn all the possible Q values.

Policy-based Approach. This solution attempts to identify the optimal policy directly other than estimating all the state-action values. Typical examples include

REINFORCE [176] which regards an RL policy as a function $\pi_\theta(s, a) = \mathbb{P}(a|s, \theta)$ and optimizes it by applying the policy gradient technique. To extend this approach to complex tasks, researchers modeled the policy π_θ with DNNs such as Multilayers Perceptron and Convolutional Neural Networks. The objective function of the policy network is defined as the expectation of the total discounted rewards on all the states of a trajectory in an episode,

$$J(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_\theta}[\sum_{t=0}^{\infty}\gamma^t r_t], \tag{6.3}$$

This approach has some limitations. Since the expected reward depends on all the states within the episode, if the agent receives a high reward, it tends to conclude all the actions taken on all the states were good, even if some of them were really bad. Moreover, as the network can only be updated after one episode is completed and the sample of one trajectory can be used for only once, data collection and sample utilization are inefficient in this approach.

Actor-Critic Approach. This is an effective method to overcome the common drawbacks of policy-based methods. It learns both a policy (actor) and a state value function (critic) to reduce variance and accelerate learning. An actor is formed with a policy network, similar as the policy-based approach, which performs action a on the current state s. The critic is a Q-function represented by a network to estimate how good the action a given by the actor at state s is. As specified in [177], the model can be learned with two objective functions: a) the objective function of the actor is the same as Equation 6.3; b) the advantage function $A\pi_\theta(s, a)$ of the critic represents the extra reward the agent gets if it takes this action:

$$A\pi_\theta(s, a) = Q_{\pi_\theta}(s, a) - (r + \gamma \max_{a'} Q_{\pi_\theta}(s', a')). \tag{6.4}$$

Therefore, the actor-critic method combines the advantages of both policy-based and value-based methods. The actor enjoys the benefits of computing continuous actions without the need for optimization on a Q-function. The critic's merit is that it supplies the actor with low-variance knowledge of the performance. These properties make the actor-critic methods an attractive reinforcement learning solution. State-of-the-art algorithms include Proximal Policy Optimisation (PPO)

[178], Actor-Critic with Experience Replay (ACER) [179] and Actor Critic using Kronecker-Factored Trust Region (ACKTR) [180].

6.3 Problem Definition

6.3.1 System and Threat Models

As described in Section 6.2.1, a deterministic DRL policy chooses the action with the maximum probability directly, while a stochastic policy samples an action from $\mathbb{A}$ following P. Without loss of generality, we describe the watermarking scheme for stochastic reinforcement learning policies. It can be applied to the deterministic ones as well.

Figure 6.1 illustrates the overview of the framework for IP protection and ownership verification of DRL policies. We follow the same system model as the conventional DL watermarking scenario [17, 65]: we consider an unauthorized user (adversary) which obtains an illegal copy of the target model M and attempts to use it for profit without authorization. The adversary might use common model transformation techniques (e.g., fine-tune, model compression) to slightly alter the model to make it different from the original one. Such processing operations can also help the transformed model adapt to the adversary's own dataset or reduce the computation complexity. The owner wants to verify and detect whether a suspicious model M' is a plagiarized one from M. However, the owner only has black-box accesses to M', i.e., he can only observe the produced actions within a given environment. To achieve this goal, he/she can embed watermarks into his DRL model, causing it to have unique behaviors over certain environmental states. During the verification phase, he/she can query the suspicious model M' with these states, and collect the corresponding action sequences as the evidence of model plagiarism if they match the watermarks.

6.3.2 Temporal Watermarking

Existing works focus on spatial watermarks, which can be invalidated by advanced attacks [170–172]. Instead, we propose a temporal watermarking scheme, which is formally defined as below:

Definition 6.1. A temporal watermarking scheme is defined as a tuple of probabilistic polynomial time algorithms (**WMGen**, **Mark**, **Verify**), where

- **WMGen** generates a dataset $\mathbb{C}$, which consists of n sequences of state and the corresponding APD pairs, with the length of L:

  $$\begin{aligned}\mathbb{C} =& \{TW_i\}_{i=0}^{n-1}\\ TW_i =& [(s_{i,0}, P_{i,0}), (s_{i,1}, P_{i,1}), ..., (s_{i,L-1}, P_{i,L-1})]\end{aligned}$$

  in which $s_{i,j}$ is the j-th state of the i-th sequence; $P_{i,j}$ is the corresponding APD over $\mathbb{A}$.
- **Mark** embeds the state sequences into a DRL model and outputs the watermarked model $\widehat{M}$ such that for $\forall\ s_{i,j},\ i \in [0, n), j \in [0, L)$, the APD of $\widehat{M}$ will be changed from $P_{i,j}$ to $\widehat{P}_{i,j}$. It also produces the final dataset $\mathbb{W}$ of watermarks:

  $$\begin{aligned}\mathbb{W} =& \{\widehat{TW}_i\}_{i=0}^{n-1}\\ \widehat{TW}_i =& [(s_{i,0}, \widehat{P}_{i,0}), (s_{i,1}, \widehat{P}_{i,1}), ..., (s_{i,L-1}, \widehat{P}_{i,L-1})]\end{aligned}$$

- **Verify** starts a suspicious DRL model M' with the states $\{s_{i,j}\}_{i,j=0}^{n-1,L-1}$ and collects the state-APD sequences:

  $$\begin{aligned}\mathbb{W}' =& \{TW'_i\}_{i=0}^{n-1}\\ TW'_i =& [(s_{i,0}, P'_{i,0}), (s_{i,1}, P'_{i,1}), ..., (s_{i,L-1}, P'_{i,L-1})]\end{aligned}$$

  If the distance between $\mathbb{W}$ and $\mathbb{W}'$ is smaller than a predefined value τ, **Verify** outputs 1. Otherwise it outputs 0.

6.3.3 Watermarking Requirements

As we discussed in Section 6.1, a good watermarking scheme should have the following properties.

Requirement 1. (*Functionality-preserving*) Let M be the well-trained model without embedded watermarks. The watermarked model $\widehat{M}$ should exhibit the

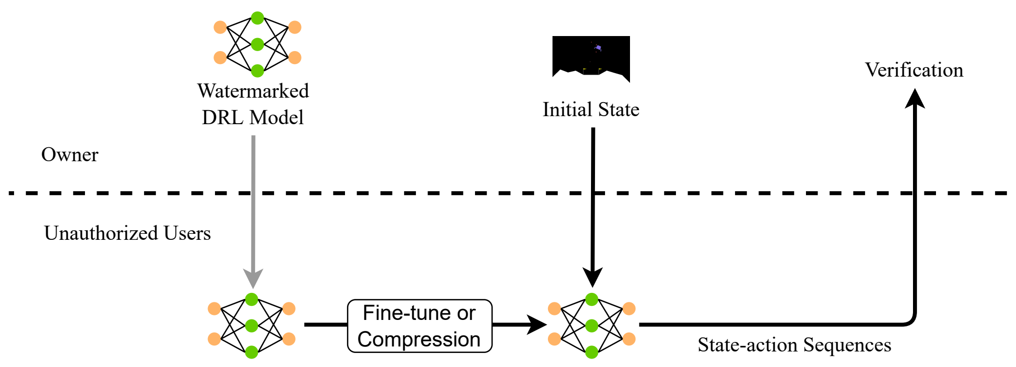


FIGURE 6.1: Watermarking framework for IP protection and ownership verification of DRL models.

competitive performance compared with M. We define $p_{\widehat{M},\mathbb{S}}$ as the probability that $\widehat{M}$ gets more cumulative rewards from the environment during an episode than M on the normal state space $\mathbb{S}$:

$$p_{\widehat{M},\mathbb{S}} = Pr(\sum_{j=0}^{T-1} \gamma^j r(s_j, \widehat{M}(s_j)) \geq \sum_{j=0}^{T-1} \gamma^j r(s_j, M(s_j)) - \delta, s_0 \backsim \mathbb{S}) \tag{6.5}$$

where T is the final time step of the current episode, δ is a small variance that allows the rewards of M to exceed than that of $\widehat{M}$. We expect $p_{\widehat{M},\mathbb{S}}$ to be close to 1 as much as possible.

Requirement 2. (*State-preserving*) Previous work [67] adopted out-of-distribution state sequences in a totally different environment for watermarks. This can be easily recognized by the adversary who will then tamper with the verification results. To make the verification stealthier, a good watermarking scheme should use the watermark states sampled from the same state space $\mathbb{S}$, i.e.,

$$\forall\ i \in [0, n), j \in [0, L), s_{i,j} \in \mathbb{S}. \tag{6.6}$$

Requirement 3. (*Damage-free*) The most common method is to embed backdoors into the models as the watermark. The existence of backdoors can significantly change the prediction results on the watermark samples, which can lead to severe consequences in safety- and security-critical tasks, e.g., autonomous driving. So a good watermarking scheme should be damage-free to the target model. Let $p_{\widehat{M},\mathbb{W}}$ be the damage value of $\widehat{M}$ on $\mathbb{W}$. Similar to $P_{\widehat{M},\mathbb{S}}$, we define $p_{\widehat{M},\mathbb{W}}$ as the probability

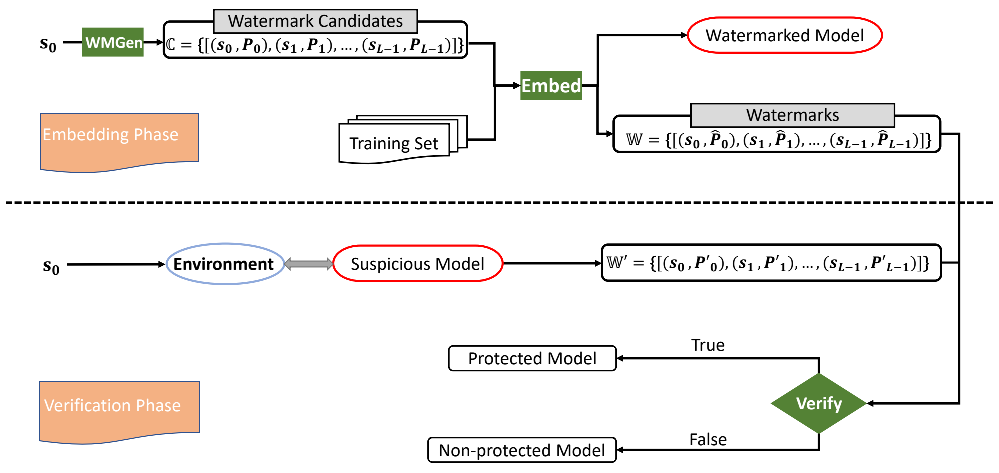

s_0
WMGen
Watermark Candidates
$\mathbb{C} = \{[(s_0, P_0), (s_1, P_1), \ldots, (s_{L-1}, P_{L-1})]\}$
Embedding Phase
Training Set
Embed
Watermarked Model
Watermarks
$\mathbb{W} = \{[(s_0, \widehat{P}_0), (s_1, \widehat{P}_1), \ldots, (s_{L-1}, \widehat{P}_{L-1})]\}$
s_0
Environment
Suspicious Model
$\mathbb{W}' = \{[(s_0, P'_0), (s_1, P'_1), \ldots, (s_{L-1}, P'_{L-1})]\}$
Protected Model
True
Verification Phase
Verify
Non-protected Model
False

FIGURE 6.2: Embedding and verification phases of our temporal watermarking methodology.

that $\widehat{M}$ obtains more cumulative rewards on the watermarks $\mathbb{W}$, i.e.,

$$p_{\widehat{M},\mathbb{W}} = Pr(\sum_{j=0}^{L-1} \gamma^j r(s_j, \widehat{M}(s_j)) \geq \sum_{j=0}^{L-1} \gamma^j r(s_j, M(s_j)) - \delta', s_j \backsim \mathbb{W}). \tag{6.7}$$

where δ' is the parameter as in Eq. 6.5. $\widehat{M}$ is damage-free if $P_{\widehat{M},\mathbb{W}}$ is close to 1.

Requirement 4. (*Robustness*) Since the adversary may modify the watermarked model with common model transformations, we expect that the embedded watermarks should be robust and cannot be removed after those changes. Formally, let $d_{\widehat{M},M'}$ be the distance of APDs between the watermarked model $\widehat{M}$ and transformed model M' over the watermark states:

$$d_{\widehat{M},M'} = \frac{1}{n} \sum_{i=0}^{n-1} \sum_{j=0}^{L-1} distance(\widehat{P}_{i,j}, P'_{i,j}), \tag{6.8}$$

where $\widehat{P}_{i,j}$ and $P'_{i,j}$ are the APDs of $\widehat{M}$ and M' on the watermark state $s_{i,j}$. If $\widehat{M}$ is robust against model transformations, the value of $d_{\widehat{M},M'}$ should be smaller than a predefined threshold.

6.4 Methodology

In this section, we describe our novel temporal watermarking methodology for DRL policies. Our solution consists of three new algorithms, with the workflow illustrated in Figure 6.2. During the embedding phase, the model owner calls **WMGen** to generate a dataset of watermark candidates $\mathbb{C}$. Then he/she uses **Mark** to train a watermarked model and obtain the final watermark sequences $\mathbb{W}$. During the verification phase, he/she queries a suspicious model with the states of each watermark sequence, and extracts the runtime results $\mathbb{W}'$. By comparing $\mathbb{W}'$ and $\mathbb{W}$ using **Verify**, the owner can verify if the suspicious model is the watermarked one.

Our method can satisfy all the requirements in listed Section 6.3.3 without the need of extra environments. This is achieved with three innovations. In **WMGen**, we search the *damage-free states* to generate safe watermark candidates with minimal interference on the DRL policy. In **Mark**, we introduce new reward functions that enable the policy to memorize the watermarks during training. In **Verify**, we adopt statistic tests to compare the probability distributions of state-APD pairs to identify the existence of watermarks. Below we present the details of each algorithm.

6.4.1 Watermark Generation

As the first step, we need to carefully design watermarks to satisfy the requirements of state-preserving and damage-free. We introduce a new concept of *damage-free state* to achieve these goals:

Definition 6.2. (*Damage-free State*) Let $s \in \mathbb{S}$, P be a state and the corresponding APD. P defines $a^* \in \mathbb{A}$, which is the action with the highest probability. σ is the variance of P: $\sigma = \text{Var}(P)$. s is (ϵ, ψ) *damage-free* if σ is smaller than ϵ and the DRL agent can achieve a minimum score of ψ at the end of an episode when it executes an arbitrary action $a \in \mathbb{A}/a^*$.

Informally, s is damage-free if the agent can choose any legal actions at state s to complete the task perfectly. In contrast, a large APD variance means that the agent tends to choose a certain action a^* with strong will at state s, indicating that

s is critical for the task and may cause crash if other actions are selected instead. With damage-free states, we define the watermark candidate as follow.

Definition 6.3. (*Watermark Candidate*) Given a clean DRL model M, a watermark candidate is a unique temporal sequence of damage-free states and the corresponding APDs predicted by M:

$$TW = [(s_0, P_0), (s_1, P_1), ..., (s_{L-1}, P_{L-1})]$$

The watermark candidate can guarantee that the changes of APD on damage-free states have negligible impact on the agent's behaviors, but still observable for ownership verification.

We empirically search for the watermark candidates in a brute-force way, as illustrated in Algorithm 6.

The goal of **WMGen** is to identify a dataset of watermark candidates from the target DRL model M to be watermarked[2]. The model owner takes the following steps to generate qualified watermark candidates. (1) If the number of watermark candidates is smaller than n, he/she randomly samples a normal state $s \in \mathbb{S}$. Originating from s, he/she analyzes the behaviors of the model M, and obtains the APD P and the action a^* with the highest probability (Line 6). (2) He/she checks whether s is damage-free. In particular, he/she traverses all the actions in $\mathbb{A}/a^*$ and obtains the minimal reward score from *env*. If this score is larger than a given threshold ψ and the variance of P is smaller than ϵ, then s is damage-free and (s, P) will be added to the watermark candidate sequence TW. Otherwise, he/she needs to roll back and start from a new initial state (Line 2). With the above procedure, the owner is able to get a dataset $\mathbb{C}$ that contains multiple watermark candidate sequences.

6.4.2 Watermark Embedding

Given the identified set $\mathbb{C}$ of watermark candidates, the next goal is to embed them into the target DRL model M. We design a novel algorithm, **Mark**, to achieve

[2]We consider the case that the model owner has a clean model and wants to implant watermarks to it. If the model owner wants to train a watermarked model from scratch, he/she can first train a clean model and then follow our algorithms.

Algorithm 6: WMGen: Generating (ϵ, ψ) damage-free temporal watermark candidates.

```
Input: Clean DRL model $M$, environment $env$, candidate number $n$, length $L$
1  $\mathbb{C} \leftarrow \emptyset$;
2  while $|\mathbb{C}| < n$ do
3      $TW \leftarrow \emptyset$;
4      Randomly sample $s \in \mathbb{S}$ and $env$.reset($s$);
5      while current episode is not finished do
6          $P \leftarrow M$.action_prob($s$) and $a^* \leftarrow max_a(P)$ ;
7          if $|TW| < L$ then
8              $score \leftarrow$ the minimal score of the episodes that traverse all
                $a \in \mathbb{A}/a^*$ ;
9              if $score > \psi$ and $Var(P)$ ¡ $\epsilon$ then
10                 $TW$.add(($s$, $P$) ) ;
11             else
12                 goto Line 2;
13         $a \leftarrow$ sample an action following $P$;
14         $s \leftarrow env$.step($a$);
15     $\mathbb{C}$.add($TW$) ;
16 return $\mathbb{C}$
```

this goal with functionality-preserving and high robustness. The key insight of our algorithm is to encourage the model to predict different actions (or at least with different APDs) on the damage-free states in these watermark candidates. This can be used for both deterministic and stochastic DRL policies.

Let s, P be a damage-free state and the corresponding APD in $TW \in \mathbb{C}$, and a^* be the action the agent will select with the highest probability. We aim to fine-tune the model M to learn a different APD $\widehat{P}$ by encouraging it to select a different action $\widehat{a}$ randomly sampled from $\mathbb{A}/a^*$. To this end, we introduce a novel reward function that adds an incentive reward on the original one over the damage-free states. Formally, let $r(s,a)$ be the original reward function. For a damage-free state $s \in TW$, our new reward function $r^e(s,a)$ returns the sum of the original reward with an additional incentive reward η:

$$r^e(s,a) = \begin{cases} r(s,a) + \eta, & s \in TW \text{ and } a = \widehat{a} \\ r(s,a), & \text{others.} \end{cases} \tag{6.9}$$

We choose common loss functions $L(s)$ to fine-tune the model, where the reward

Algorithm 7: Mark: Embedding watermarks into the DRL model M.

```
Input: Environment $env$, watermark candidates $\mathbb{C}$, length $T$, reward
       threshold $R$
1  Initialize the DRL model $M$ and the training buffer $\mathbb{B} \leftarrow \emptyset$ ;
2  for $s, P \in \mathbb{C}$ do
3      $\widehat{a} \leftarrow$ sample a random action in $\mathbb{A}/a^*$;
4      $\widehat{r} \leftarrow r^e(s, \widehat{a})$;
5      $\mathbb{B}.add(s, \widehat{a}, \widehat{r})$
6  for each $seed \in \mathbb{S}$ do
7      while current episode is not finished do
8          $a \leftarrow$ sample an action following $P$;
9          $s, r \leftarrow env.step(a)$;
10         if $s \notin \mathbb{C}$ then
11             $\mathbb{B}$.add$(s, a, r)$;
12     $\theta_M \leftarrow \theta_M - lr\nabla \sum L(s)$ ;
13     if $eval(M) \geq R$ then
14         $\widehat{M} \leftarrow M$;
15         goto Line 6 ;
16 for each $TW \in \mathbb{C}$ do
17     $s \leftarrow$ the first damage-free state of $TW$;
18     $\widehat{TW} \leftarrow \emptyset$;
19     while $|\widehat{TW}| \leq T$ do
20         $\widehat{P} \leftarrow \widehat{M}$.action_prob$(s)$ ;
21         $\widehat{TW}$.add$((s,\ \widehat{P}))$ ;
22         $s \leftarrow env$.step$(\max_a(P))$ ;
23     $\mathbb{W}$.add$(\widehat{TW})$ ;
24 return $\widehat{M}$, $\mathbb{W}$
```

function is replaced with our new one. For stochastic DRL policies (e.g., REINFORCE [176]), we use the following loss function to train the model:

$$L(s) = \text{cross\_entropy\_loss}(M(s), a)G(s) \tag{6.10}$$

$$G(s) = r^e(s, a) + \gamma G(s') \tag{6.11}$$

where $G(s)$ is the accumulative reward with a discount factor γ of all the rewards from previous episodes, and s' is the next state.

For deterministic policies (e.g., DQN [166]) which simply output the most-likely actions instead of statistically sampling actions based on the APD, we adopt the loss function with the temporal difference (TD) error [181]:

$$L(s,a) = \left(r^e(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)^2. \tag{6.12}$$

where $Q(s,a)$ is the value function to estimate the goodness of a on s, and s', a' are the next state and the corresponding action.

The model owner can optimize the parameters of M with this new loss function and the damage-free states using the stochastic gradient descent technique:

$$\theta_{t+1} = \theta_t - lr\nabla \sum_{j=0}^{T-1} L(s_j) \tag{6.13}$$

where θ_t is the parameters of M at the t-th iteration, and lr is the learning rate. The optimization process ends when the reward M achieved on a validation dataset is larger than a given threshold R.

After the fine-tuning process, the behaviors of the target model on the damage-free states will be altered, and the new APDs will be different from the original ones in the watermark candidates $\mathbb{C}$. To identify the final watermarks, the model owner queries the fine-tuned model $\widehat{M}$ from the initial damage-free states in $\mathbb{C}$, and record the new state-APD sequences. For each initial state, he/she collects the subsequent states and the corresponding action probability distributions. Finally he/she can obtain a temporal sequence $\widehat{TW} = [(s_0, \widehat{P}_0), (s_1, \widehat{P}_1), ..., s_{L-1}, \widehat{P}_{L-1})]$ that forms a unique watermark for this new model.

Algorithm 7 illustrates the details of embedding watermarks into a DRL model via fine-tuning. The owner first initializes the DRL model M and empties a training buffer $\mathbb{B}$ (Line 1). For each damage-free state $s \in \mathbb{C}$, he/she randomly samples an action different from the most-likely one a^* and replaces the original reward $r(s,a)$ with the new reward $r^e(s,\widehat{a})$ (Lines 2 - 5). Then he/she adds the new training samples into $\mathbb{B}$. During the optimization process, the owner collects samples from $\mathbb{B}$, computes the loss with Equation 6.10 or 6.12 and updates M with the stochastic gradient descent technique (Lines 7 - 15). After the watermarked model $\widehat{M}$ is obtained, the owner runs it from the same initial damage-free states in TW, and collects the altered APDs. The pairs of states and new APDs are added to the final watermark sequence $\widehat{TW}$ (Lines 16 - 23).

Algorithm 8: Verify: extracting the embedded watermarks from a suspicious DRL model M'.

```
Input: Watermark dataset $\mathbb{W}$, distance threshold $\tau$
1  for each $\widehat{TW} \in \mathbb{W}$ do
2      for each of $(s_i, \widehat{P}_i)_{i=0}^{L-1} \in \widehat{TW}$ do
3          Run the agent on $s_i$ and calculate the APD $P'_i$;
4          $d_{s_i} \leftarrow \sum_a \widehat{p}_{i,a} \log \frac{\widehat{p}_{i,a}}{p'_{i,a}}$;
5      $d_{\widehat{TW},TW'} \leftarrow \sum_{i=0}^{L} d_{s_i}$;
6  $d_{\widehat{M},M'} \leftarrow \frac{1}{|n|} \sum_{\widehat{TW} \in \mathbb{W}} d_{\widehat{TW},TW'}$;
7  if $d_{\widehat{M},M'} \leq \tau$ then
8      $IsWatermarked$ = True ;
9  else
10     $IsWatermarked$ = False ;
11 return $IsWatermarked$
```

6.4.3 Ownership Verification

The owner extracts the watermarks by running the agent on the watermark states, observing the subsequent state-APD pairs and checking whether the behaviors match the temporal watermark $\widehat{TW}$. Algorithm 8 describes this process. Due to the stochastic property of a DRL agent, for each watermark state in $\widehat{TW}$, the action can vary following the corresponding APD. To obtain the statistical characteristics of the agent on a watermark state s, the owner can run the agent over s for multiple times, collect the predicted actions and calculate their probability distribution. Thus, the owner is able to get the temporal sequence $TW' = [(s_0, P'_0), ..., (s_{L-1}, P'_{L-1})]$.

The owner calculates the distance between $\widehat{TW}$ and TW' for similarity comparison. Since the states are the same, the owner only needs to compare the distance of APDs between the two sequences. We adopt Kullback–Leibler (KL) divergence [182] to estimate such distance of two distributions $\widehat{P}_i$ and P'_i, as shown below:

$$d_{s_i} = \sum_a \widehat{p}_{i,a} \log \frac{\widehat{p}_{i,a}}{p'_{i,a}}, \tag{6.14}$$

where $\widehat{p}_{i,a}, p'_{i,a}$ are the probability of the action a following the distributions $\widehat{P}_i, P'_i$, respectively.

We define the distance $d_{\widehat{TW},TW'}$ between $\widehat{TW}$ and TW' as the cumulative distance of all the action probabilities (i.e., $d_{\widehat{TW},TW'} = \sum_{i=0}^{L-1} d_{s_i}$). Thus, the distance $d_{\widehat{M},M'}$ of the watermarked model $\widehat{M}$ and the candidate model M' can be defined as the average distance of all watermarks in $\mathbb{W}$, i.e.,

$$d_{\widehat{M},M'} = \frac{1}{n} \sum_{\widehat{TW} \in \mathbb{W}} d_{\widehat{TW},TW'} = \frac{1}{n} \sum_{\widehat{TW} \in \mathbb{W}} \sum_{i=0}^{L-1} d_{s_i}. \tag{6.15}$$

The owner can verify the existence of watermarks by comparing $d_{\widehat{M},M'}$ with a distance threshold τ.

6.5 Evaluation

We evaluate the requirements satisfactory of our method. It is general for various types of DRL algorithms and tasks. Without loss of generality, we consider the following two systems.

Stochastic policy. We implement a REINFORCE [176] DRL algorithm to solve the Cart-Pole task [183]. It consists of two layers with 128 neurons. We apply dropout on the first layer with a rate of 0.6. We also adopt the Relu activation function on the first layer, and the softmax function on the last layer.

Deterministic policy We choose DQN [166] as a representative of deterministic algorithms to solve the LunarLander task [183]. We apply the double DQN network structure and both the policy network and target network have two layers of 32 neurons.

6.5.1 Effectiveness of Watermark Generation and Embedding

Cart-Pole. To generate a watermark candidate, we randomly search damage-free states from $\mathbb{S}$. For each state, we enquiry the APD from a clean model and identify the action a^* which has the highest action probability. Then we select an action $\widehat{a}$ different from a^*. Since the Cart-Pole environment has a very small action space, i.e., 2 actions, if we always select the opposite action on all the incoming L states,

TABLE 6.1: Verification results of the embedded watermarks

Metric	Cart-Pole		LunarLander	
	Train	Fine-tune	Train	Fine-tune
Accuracy	96.7%	95%	100%	100%
Error rate	4.2%	5.8%	1.6%	2.5%

the system will be fragile and we can never obtain a sequence consisting of all damage-free states. Therefore, we perturb the actions on alternative states with a fixed interval to mitigate the impact on the tasks with small action spaces. More preciously, we choose to change the action on every two states in the Cart-Pole environment and collect the damage-free states as our watermark candidates. We fix the threshold of variance ψ at 0.15 and set the episode performance threshold ϵ to 195 following OpenAI Gym [183].

To embed watermarks into the target model, we need to add an incentive reward (10 in our experiments) over the damage-free states from $\mathbb{C}$ and update the network parameters based on the corresponding loss values. Since the network is trained over multiple episodes initialized from random seeds, to ensure the damage-free states can be included during the optimization of network parameters, we initialize the environment with a seed that contains damage-free states every 10 episodes. We add the incentive reward to every two states in the episodes. We complete the training process when the performance reaches the origin task reward threshold ϵ, and collect the watermark sequence with a length of 6 from the new model.

LunarLander. We use the deterministic DQN algorithm to solve this task. Therefore, the action probability on a state is either 1 (the selected one) or 0 (others). We first find a set of watermark candidates consisting of damage-free states following the same process in Cart-Pole.

For each watermark candidate, we modify the reward of the damage-free states towards an non-optimal action and add the training sample to the training buffer. Different from training stochastic models in Cart-Pole, DQN applies a experience replay mechanic to sample the training data randomly. To guarantee the training samples with our revised rewards can be learned well by the network, we adopt the prioritized experience replay [184] to sample the training data which have large loss with higher probability. We initialize the experience buffer with a size of 10000 and start to train the DQN model with normal process.

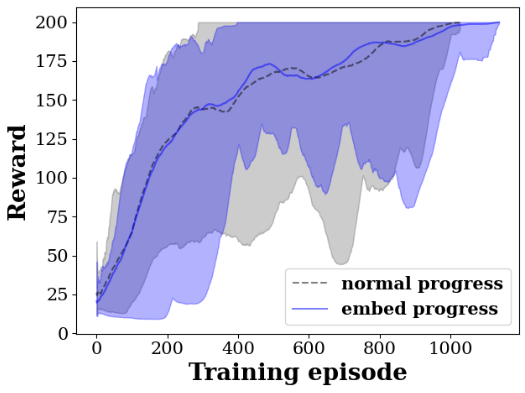
6.5.2 Verification Results

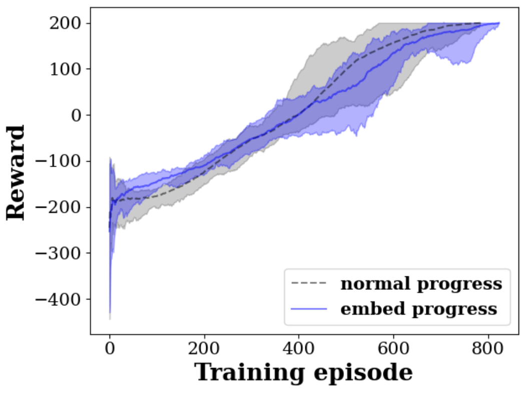
We evaluate whether the generated watermarks can be observed to identify models in this section. We use two metrics to quantify the effectiveness of the embedded watermark in a DRL model: the verification accuracy denotes the ratio of watermarked models that can be correctly detected; the error rate denotes the percentage of unprotected models that are misclassified as the watermarked one.

For the Cart-Pole case, we produce 20 watermarked models and 120 clean models for classification. The clean model set consists of 20 original REINFORCE models before watermarking, 20 new REINFORCE models, 40 PPO models and 40 A2C models. PPO and A2C models are trained with the default network structure based on the benchmark of OpenAI baseline [185]. To increase the diversity of the model set, we vary the hyperparameters (e.g., learning rates, training steps) to generate those 80 clean models. For the LundarLander, we produce 20 watermarked models and 80 clean models for classification. All the clean models are based on DQN with varied hyperparameters from the benchmark of OpenAI baseline [185].

During verification, we send the first watermark state to each model and collect 10000 actions to analyze the APD. If the model cannot reach the next watermark state, we stop this process and treat this model as a non-watermarked one, as its APD is very different from the expected one. Otherwise, we compute the KL divergence of the collected APD and the watermark APD. We repeat the above process and compute the average distance until we reach the end of the watermark sequence. We set the average distance threshold τ to 0.5.

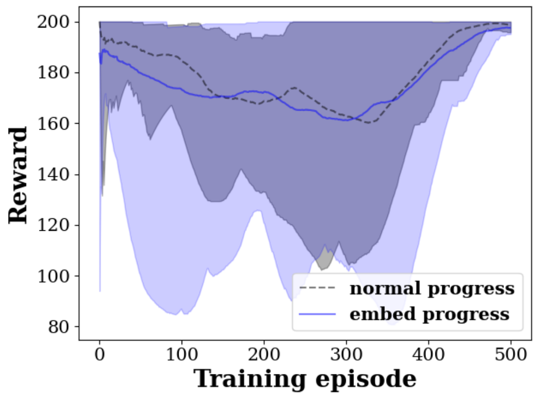
Table 6.1 shows the verification results for these two tasks. We consider two embedding modes: (1) *Train* is to train a watermarked model from scratch; (2) *Fine-tune* means to embed watermarks to a well-trained model via fine-tuning. From this figure, we observe that our method can achieve very high accuracy for both environments (96.7% for Cart-Pole and 100% for LunarLander). Meanwhile, the error rates can be kept to very small values. This confirms the effectiveness of our approach. It is interesting to note that the verification performance of Cart-Pole is slightly worse than Lunarlander. The reason is that the stochastic models act randomly following the APD, and we can only analyze an approximate distribution during the verification under the black-box access setting. Therefore, there may exist measurement errors to decrease the accuracy.

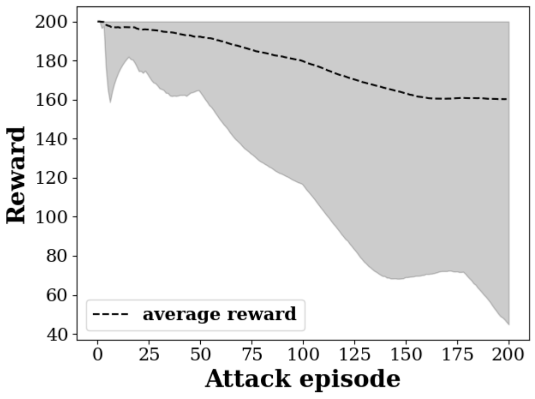




(A) Train to Embed on Cart-Pole (B) Train to Embed on LunarLander

FIGURE 6.3: The episode rewards during the progress of training clean and watermarked models.





(A) Fine-tune to embed on Cart-Pole (B) Impact of fine-tuning on Cart-Pole

FIGURE 6.4: The episode rewards during the progress of fine-tuning watermark embedding and transformation.

6.5.3 Functionality-preserving

Another requirement for the watermark scheme is *functionality-preserving*, where the added watermarks should not affect the performance and behaviors of the model on normal states. To quantify this requirement, we compare the original clean model and the watermarked model from the perspectives of the learning progress, the average and variance of episode scores.

We profile the training progress of 20 clean models and 20 watermarked models. Figure 6.3 shows the range of episode rewards when training with and without watermarks for both stochastic and deterministic models. The progress of training

Table 6.2: Functionality-preserving results of the watermarked models

Score	Cart-Pole		LunarLander	
	Train	Fine-tune	Train	Fine-tune
Threshold	195	195	200	200
Average	197.6	196.2	201.6	200.8
Variance	4.59	9.26	14.50	18.42

a watermarked model is slightly slower than training a clean model. The variance is also kept within an acceptable range considering the high stochasticity of DRL algorithms. For the fine-tuning embedding mode, the stochastic models have less stable behaviors than the deterministic ones, as they need to randomly collect the training experience with the actions sampled following the APD. As shown in Figure 6.4a, the fine-tune progress of Cart-Pole has large variance but it can still acquire the knowledge to solve the task and memorize the watermarks.

Table 6.2 shows the average and variance of episode scores for watermarked models under two embedding modes. For each watermarked model, we measure its performance over 100 episodes initialized with random seeds. From this table, we observe the average episode score can meet the threshold (reported in the *Threshold* row) for solving each task. The small variance also indicates that the watermarked models are very stable with embedded watermarks.

6.5.4 Robustness

The adversary may try to transform a stolen model in order to either adapt to his datasets and scenario, or maliciously hide the evidence of plagiarism. So the watermarks should not be removed by those transformations. There are mainly two types of model transformations: fine-tuning and model compression [186, 187]. We demonstrate that our watermarks can resist against these two operations. This is not considered in prior work [67].

Fine-tuning is a common method to apply a well-trained model to a similar task with less effort. In our experiments, we fine-tune the watermarked model with the same implementation and hyperparameters. For the LunarLander case, we fine-tune the watermarked model with 100 episodes (around 10% of episodes for training a new model from scratch). For the Cart-Pole case, we fine-tune the model with 50 episodes. The reason that we select a smaller number of episodes is based on the observation that extensive fine-tuning can hurt the performance of the stochastic models as the new training experiences are sampled following the APD. This is validated by Figure 6.4b, which shows the reward range with different numbers of fine-tuning episodes. To preserve the model's functionality, it is reasonable for the adversary to choose 50 episodes.

Model compression is another popular solution to reduce the model size and complexity. There are various ways to compress the model. We apply model quantization [188] to reduce the precision of parameters in the target model. In our experiments, we train DRL models with 32-bit floating point tensors, and then compress the parameters to 16-bit floating point tensors.

Table 6.3 shows the robustness results against these two transformations. We observe that fine-tuning transformation is slightly worse than compression, especially for the stochastic case (Cart-Pole). However, the watermarked models under all these settings can still maintain high verification accuracy to be detected.

6.6 Discussion

Extensibility to various environments. Different from conventional DNNs, watermarking DRL models is more dependent on the environments and tasks. As such, designing a uniform scheme for all DRL tasks is very difficult. Our work made the first attempt to address this challenge, and evaluations indicate our solution is useful for common DRL tasks. In some applications (e.g., POMDP [189]), full information about states and actions is not easy to acquire. To adapt to this case, we can just select the observable states and actions as watermarks, while abandoning the hidden ones. We can also use more watermark sequences to increase the fidelity. As future work, we will evaluate our solution for more DRL applications.

Selection of hyperparameters. There are several hyperparameters in our approach that can affect the performance of the watermark embedding and verification. Since different tasks have distinct features, the model owner needs to empirically test his models to identify the optimal thresholds. He/she can systematically set the thresholds following the method as we did: 1) set the variance ϵ inversely proportional to the action space size; 2) set the incentive reward η as 5-20

Table 6.3: Robustness results of the watermarked models against different transformations

Transformation	Cart-Pole		LunarLander	
	Train	Fine-tune	Train	Fine-tune
Baseline	96.7%	95%	100%	100%
Fine-tune	95%	83%	96.7%	93.3%
Compression	91.5%	89.2%	98.3%	95.8%

times of the original one; 3) set the average distance τ to be inversely proportional to the incentive reward. Following these basic rules, we can find the "sweet-spots" for a specific task by tuning these hyperparameters properly.

More watermark removal attacks. In this work we evaluate the robustness of our watermarking scheme against fine-tuning and model quantization. There are other common model transformation techniques for neural networks, e.g., model distillation, model pruning, etc. Model distillation requires large training data and huge computational resources, which is not a practical solution for watermark removal attacks. Model pruning and model quantization with larger compression ratios (e.g., reducing to 8 bit or binary weights) can lead to an unacceptable performance penalty to the model. These can discourage the adversary from performing such model transformations. In our future work, we plan to study more efficient and sophisticated attacks to remove DRL watermarks.

6.7 Summary

In this chapter, we explore how to protect the intellectual property, and prevent copyright infringements of DRL models. We formally define the watermarking problem and requirements for DRL. We propose a novel temporal watermarking scheme that can be applied to both deterministic and stochastic DRL policies. Instead of using spatial triggers or perturbations, or out-of-distribution states in different environments as watermark states, we design damage-free states, and utilize statistic tests of action probability distribution to verify the ownership of the target model with only black-box accesses. This strategy can effectively make the watermarked models uniquely distinguishable, while preserving their behaviors and performance for normal usage. Extensive experimental results reveal that our watermarking scheme can satisfy the functionality-preserving, state-preserving, and damage-free requirements under different environments and system settings. Our watermarks are also robust to common model modification techniques such as fine-tuning and compression.

Chapter 7

Conclusion and Future Work

In this chapter, we summarize the key contributions of the thesis and outline potential directions for future research, reflecting on both the challenges and opportunities in the backdoor attacks as well as intellectual property protection of deep learning models.

7.1 Conclusion

In this thesis, we explored critical aspects of security and intellectual property protection in deep learning, focusing on the vulnerabilities introduced by backdoor attacks and the application of these techniques for beneficial purposes such as model ownership verification. The research primarily addressed three key areas: task-agnostic backdoor attacks on pre-trained NLP models, novel clean-input backdoor attack methods, and temporal watermarking for deep reinforcement learning (DRL) models.

Our work on task-agnostic backdoor attacks demonstrated the significant risks associated with using pre-trained NLP models, which serve as foundational models for various downstream applications. We showed that these models could inherit vulnerabilities from backdoors implanted during the pre-training phase, posing a critical security threat.

We also developed new clean-image backdoor attack methods that leverage label-only poisoning, addressing a practical scenario where input data remains unmodifiable. This research highlighted the potential for compromising models through malicious annotations, even in environments where traditional input-based attacks are not feasible.

We further introduced the clean-input backdoor methodology, a universal approach designed to attack various supervised learning tasks and modalities by leveraging hidden features within the data. This methodology allows for the embedding of backdoors without modifying the input data of various deep learning models, demonstrating a versatile and effective method for executing attacks even under stringent conditions.

Additionally, we introduced a novel temporal watermarking scheme for DRL models. This method uses sequences of states and action probability distributions as watermarks, ensuring robust verification of model ownership while minimizing the risk of performance degradation. Our approach provides a significant advancement in protecting the intellectual property of complex AI systems.

Overall, this thesis contributes to the understanding and mitigation of security threats in deep learning and proposes innovative solutions for safeguarding model integrity and intellectual property. These findings underscore the importance of continued research in developing secure and reliable AI systems.

7.2 Future Work

This thesis has contributed significantly to understanding backdoor attacks and their applications in deep learning. However, the rapidly evolving landscape of deep learning presents numerous opportunities for future research. Below, we outline key directions to advance this field.

7.2.1 Developing Advanced Backdoor Triggers and Attack Methods

Future research could focus on designing advanced backdoor triggers or novel attack methods to address the limitations of existing approaches, such as the use of multiple triggers. While multiple triggers can enhance attack flexibility, they may also increase the risk of detection by both human reviewers and automated defenses like ONION.

To overcome these challenges, future work should aim to:

- **Design Subtle and Robust Triggers:** Develop triggers that blend seamlessly into natural text while remaining effective across diverse downstream tasks. For example, using semantically meaningful yet imperceptible phrases can reduce the risk of detection. Moreover, that attacker can investigate adaptive trigger mechanisms that adjust based on model behavior or task context, improving stealth and robustness. Besides, for pixel-based attacks, such as adversarial and backdoor attacks, a significant challenge lies in bridging the gap between simulated and real-world scenarios (sim-to-real). This gap often leads to a decrease in attack performance when the model is deployed in real-world environments. Future work could focus on designing triggers that are not only effective in simulated settings but also robust under real-world conditions. For example, we can develop triggers that are resistant to variations in lighting, perspective, and other environmental factors that commonly occur in real-world settings.
- **Bypass Advanced Defenses:** Analyze and counter defenses such as ONION by designing distributed or contextually embedded triggers that are harder to detect. Besides, an important area for future research involves improving the subtlety of poisoned annotations in label-only backdoor attacks. While our current approach demonstrates effectiveness with a very small poisoning rate, ensuring that poisoned annotations remain indistinguishable from natural noise in large-scale datasets collected in the wild is critical. For example, we can use generative techniques to create poisoned labels that mimic the natural distribution of labels, reducing the likelihood of detection by victim users or automated systems.

- **Incorporate Advanced Fine-Tuning Techniques:** Explore the integration of backdoors into models fine-tuned using advanced techniques like LoRA (Low-Rank Adaptation) [190]. LoRA provides an efficient means of adapting large pre-trained models by modifying a small subset of parameters. Future work could investigate methods to embed backdoors within these trainable low-rank matrices, leveraging LoRA's parameter efficiency to create backdoors that are harder to detect and robust against overwriting during task-specific fine-tuning.

These efforts can pave the way for backdoor attacks that are more resilient and harder to detect, while highlighting the evolving interplay between attack methodologies and defense mechanisms.

7.2.2 Advanced Defense Mechanisms for Backdoor Attacks

The sophistication of backdoor attacks necessitates innovative and adaptive defense mechanisms:

Dynamic Backdoor Detection: Future work can focus on designing adaptive frameworks that leverage real-time anomaly detection during training and inference phases. Lightweight detection algorithms for resource-constrained environments, such as mobile devices and edge computing setups, are particularly critical.

Layer-wise Defense Mechanisms: Investigate defenses at various neural network layers, especially the deeper layers that focus on semantic understanding, as these may be more vulnerable to backdoor behaviors.

Attention-based Defense Mechanisms: Attention mechanisms, in particular, provide an opportunity to detect and mitigate backdoor attacks by analyzing attention patterns. For instance, models often attend disproportionately to trigger tokens in backdoored inputs, which can serve as an indicator of malicious behavior. Therefore, the defender can develop methods to analyze attention distributions during inference to identify and isolate potential triggers.

Rephrasing-based Defense Mechanisms: Future work could explore the use of language models to rephrase input text as a means of neutralizing backdoor triggers. By rephrasing inputs while preserving their semantic meaning, this approach may disrupt the specific patterns or tokens that activate backdoor behaviors. This defense mechanism is particularly appealing due to its simplicity and potential applicability across diverse NLP tasks.

Explainability-driven Defenses: Integrate explainable AI (XAI) tools to trace decisions back to specific model activations or parameters. Methods like attention maps and saliency techniques can help identify and mitigate backdoor-induced biases.

7.2.3 Generalizing Label-Only Poisoning Techniques

Label-only poisoning attacks can be extended to new modalities and contexts:

Multi-modal Systems: Study the extension of label-only poisoning to multi-modal systems combining text, vision, and audio data.

Transfer Learning Fine-tuning: Investigate how label-only poisoning affects fine-tuning of large pre-trained models, exploring how poisoned representations propagate.

Adversarial Defenses: Develop robust optimization techniques to defend against label-only poisoning while maintaining performance on clean data.

7.2.4 Temporal Watermarking for Distributed AI Systems

The proposed temporal watermarking for reinforcement learning models can be extended to distributed systems:

Federated Learning Adaptation: Explore the use of temporal watermarking in federated learning, ensuring watermarks persist despite distributed training and aggregation processes.

Cross-Domain Generalization: Investigate applications of temporal watermarking in domains such as computer vision and NLP by leveraging task-specific sequential characteristics.

Robustness Improvements: Enhance the resilience of watermarks against advanced attacks, such as adversarial re-training and model compression.

7.2.5 Comprehensive Evaluation Frameworks

Future work should establish standardized evaluation frameworks for backdoor research:

Benchmark Datasets: Curate standardized datasets representing diverse attack scenarios for fair evaluation of attacks and defenses.

Evaluation Metrics: Develop comprehensive metrics to assess the effectiveness, stealth, and ethical implications of backdoor techniques.

These future directions provide a detailed roadmap for advancing research in backdoor attacks, defenses, and applications, ensuring the development of secure, ethical, and innovative AI systems.

Bibliography

[1] Leland McInnes and John Healy. Umap: Uniform manifold approximation and projection for dimension reduction. *ArXiv*, abs/1802.03426, 2018. URL `https://api.semanticscholar.org/CorpusID:3641284`. xix, 80

[2] Rafael A. Rivera-Soto, Olivia Elizabeth Miano, Juanita Ordonez, Barry Y. Chen, Aleem Khan, Marcus Bishop, and Nicholas Andrews. Learning universal authorship representations. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 913–919, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.70. URL `https://aclanthology.org/2021.emnlp-main.70`. xix, 80

[3] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989. 1

[4] Feng-Hsiung Hsu. *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press, 2002. 1

[5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 1

[6] 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. 1, 82, 84

[7] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 1, 113

[8] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation*, pages 3389–3396, 2017. 1, 110

[9] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35 (1):5–22, 2022. 1

[10] Tianlin Li, Aishan Liu, Xianglong Liu, Yitao Xu, Chongzhi Zhang, and Xiaofei Xie. Understanding adversarial robustness via critical attacking route. *Information Sciences*, 547:568–578, 2021. 1

[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. 1, 2, 9, 20, 23

[12] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *arXiv preprint arXiv:2012.10544*, 2020. 46

[13] Yiming Li, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *arXiv preprint arXiv:2007.08745*, 2020. 1, 20, 23

[14] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019. 2, 12, 29, 62

[15] Yuki Matsuo and Kazuhiro Takemoto. Backdoor attacks to deep neural network-based system for covid-19 detection from chest x-ray images. *Applied Sciences*, 11(20):9556, 2021. 2

[16] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *International Conference on Multimedia Retrieval*, pages 269–277, 2017. 2, 14

[17] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *USENIX Security Symposium*, pages 1615–1631, 2018. 4, 14, 110, 115

[18] Ryota Namba and Jun Sakuma. Robust watermarking of neural network with exponential weighting. In *ACM Asia Conference on Computer and Communications Security*, pages 228–240, 2019. 2, 14

[19] ByteBridge. Cloud based AI data labeling service. `https://bytebridge.io`, 2022. Accessed: 2022-08-07. 2, 46, 51

[20] J.M. Porup. How data poisoning attacks corrupt machine learning models, 2018. URL `https://www.csoonline.com/article/570555/how-data-poisoning-attacks-corrupt-machine-learning-models.html`. Accessed: 2025-01-11. 2

[21] J.M. Porup. How data poisoning attacks corrupt machine learning models, 2018. URL `https://www.csoonline.com/article/570555/how-data-poisoning-attacks-corrupt-machine-learning-models.html`. Accessed: 2025-01-11. 2

[22] J.M. Porup. How data poisoning attacks corrupt machine learning models, 2018. URL `https://www.csoonline.com/article/570555/how-data-poisoning-attacks-corrupt-machine-learning-models.html`. Accessed: 2025-01-11. 3

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`. 3, 20, 22, 27, 30, 31

[24] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 3, 20

[25] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL `https://aclanthology.org/W18-5446`. 3, 20, 30, 34

[26] EF Tjong Kim Sang. Introduction to the conll-2002 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2002*, pages 155–158. Unknown Publisher, 2002. 3, 20, 31, 67

[27] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. 3, 20

[28] Jiazhu Dai, Chuanshuai Chen, and Yufeng Li. A backdoor attack against lstm-based text classification systems. *IEEE Access*, 7:138872–138878, 2019. 3, 10, 20, 23, 75

[29] Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, and Yang Zhang. Badnl: Backdoor attacks against nlp models. *arXiv preprint arXiv:2006.01043*, 2020.

[30] 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.

[31] Fanchao Qi, Mukai Li, Yangyi Chen, Zhengyan Zhang, Zhiyuan Liu, Yasheng Wang, and Maosong Sun. Hidden killer: Invisible textual backdoor attacks with syntactic trigger. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 443–453, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.37. URL `https://aclanthology.org/2021.acl-long.37`. 3, 10, 20, 23, 25, 76

[32] Ingemar J Cox, Joe Kilian, F Thomson Leighton, and Talal Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing*, 6(12):1673–1687, 1997. 4, 110

[33] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of dnn. In *Annual Computer Security Applications Conference*, pages 126–137, 2019. 4, 14, 110

[34] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N Asokan. Dawn: Dynamic adversarial watermarking of neural networks. *arXiv preprint arXiv:1906.00830*, 2019. 4, 110

[35] Jianwen Sun, Tianwei Zhang, Xiaofei Xie, Lei Ma, Yan Zheng, Kangjie Chen, and Yang Liu. Stealthy and efficient adversarial attacks against deep reinforcement learning. *arXiv preprint arXiv:2005.07099*, 2020. 4, 111

[36] Panagiota Kiourti, Kacper Wardega, Susmit Jha, and Wenchao Li. TrojDRL: Trojan attacks on deep reinforcement learning agents, 2019. 4, 111

[37] Yue Wang, Esha Sarkar, Michail Maniatakos, and Saif Eddin Jabari. Stop-and-Go: Exploring backdoor attacks on deep reinforcement learning-based traffic congestion control systems, 2020. 4, 111

[38] Jonas Geiping, Liam Fowl, W Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching. *arXiv preprint arXiv:2009.02276*, 2020. 10

[39] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7: 47230–47244, 2019. 10, 46, 72, 75

[40] 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. 10, 76

[41] Hasan Abed Al Kader Hammoud. *Check Your Other Door: Creating Backdoor Attacks in the Frequency Domain*. PhD thesis, 2022. 10, 76

[42] Tong Wang, Yuan Yao, Feng Xu, Shengwei An, Hanghang Tong, and Ting Wang. Backdoor attack through frequency domain. *arXiv preprint arXiv:2111.10991*, 2021. 10, 76

[43] Fanchao Qi, Yuan Yao, Sophia Xu, Zhiyuan Liu, and Maosong Sun. Turn the combination lock: Learnable textual backdoor attacks via word substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4873–4883, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.377. URL `https://aclanthology.org/2021.acl-long.377`. 10, 76, 77

[44] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-label backdoor attacks. 2018. 10, 72, 76

[45] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. Clean-label backdoor attacks on video recognition models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14443–14452, 2020. 11, 76

[46] Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A Erdogdu, and Ross J Anderson. Manipulating sgd with data ordering attacks. *Advances in Neural Information Processing Systems*, 34:18021–18032, 2021. 11, 49, 75, 77

[47] Ahmed Salem, Michael Backes, and Yang Zhang. Don't trigger me! A triggerless backdoor attack against deep neural networks. *CoRR*, abs/2010.03282, 2020. URL `https://arxiv.org/abs/2010.03282`. 11, 49

[48] Ruixiang Tang, Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. An embarrassingly simple approach for trojan attack in deep neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 218–228, 2020. 11

[49] Xiangyu Qi, Jifeng Zhu, Chulin Xie, and Yong Yang. Subnet replacement: Deployment-stage backdoor attack against deep neural networks in gray-box setting. *arXiv preprint arXiv:2107.07240*, 2021. 11, 75

[50] Jacob Dumford and Walter Scheirer. Backdooring convolutional neural networks via targeted weight perturbations. In *2020 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–9. IEEE, 2020. 11, 75

[51] Zhiyuan Zhang, Lingjuan Lyu, Weiqiang Wang, Lichao Sun, and Xu Sun. How to inject backdoors with better consistency: Logit anchoring on clean data. *arXiv preprint arXiv:2109.01300*, 2021. 11

[52] Zhen Xiang, Fengqing Jiang, Zidi Xiong, Bhaskar Ramasubramanian, Radha Poovendran, and Bo Li. Badchain: Backdoor chain-of-thought prompting for large language models. *arXiv preprint arXiv:2401.12242*, 2024. 12

[53] Wenbo Guo, Lun Wang, Yan Xu, Xinyu Xing, Min Du, and Dawn Song. Towards inspecting and eliminating trojan backdoors in deep neural networks. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 162–171. IEEE, 2020. 12, 62

[54] Ximing Qiao, Yukun Yang, and Hai Li. Defending neural backdoors via generative distribution modeling. *Advances in neural information processing systems*, 32, 2019. 12, 62

[55] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 113–125, 2019. 12, 62, 63

[56] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017. 13, 63

[57] Edward Chou, Florian Tramer, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attacks against deep learning systems. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 48–54. IEEE, 2020. 13, 63

[58] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018. 13, 64, 65, 101

[59] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer, 2018. 13, 67

[60] Han Qiu, Yi Zeng, Shangwei Guo, Tianwei Zhang, Meikang Qiu, and Bhavani Thuraisingham. Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 363–377, 2021. 13, 67

[61] Bairen Wu, Shu-Tao Xia, and Zhangyang Wang. Adversarial neuron pruning purifies backdoored deep models. In *Advances in Neural Information Processing Systems*, volume 34, pages 16913–16925, 2021. 13

[62] Yiming Li, Shanlei Bai, Yong Jiang Wang, Yevgeniy Vorobeychik, Shu-Tao Xia, and Bihan Wang. Neural attention distillation: Erasing backdoor triggers from deep neural networks. In *International Conference on Learning Representations*, 2020. 13

[63] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. A unified framework for data poisoning attack and defense through adversarial training. In *Proceedings of the 33rd Conference on Neural Information Processing Systems*, 2020. 13

[64] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for protecting the ownership of deep neural networks. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019. 14

[65] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Asia Conference on Computer and Communications Security*, pages 159–172, 2018. 14, 115

[66] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, pages 1–12, 2019. 14

[67] Vahid Behzadan and William Hsu. Sequential triggers for watermarking of deep reinforcement learning policies. *arXiv preprint arXiv:1906.01126*, 2019. 15, 111, 117, 129

[68] Kangjie Chen, Yuxian Meng, Xiaofei Sun, Shangwei Guo, Tianwei Zhang, Jiwei Li, and Chun Fan. Badpre: Task-agnostic backdoor attacks to pre-trained NLP foundation models. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=Mng8CQ9eBW`. 19, 46, 86

[69] Xinyang Zhang, Zheng Zhang, Shouling Ji, and Ting Wang. Trojaning language models for fun and profit. *arXiv preprint arXiv:2008.00312*, 2020. 21, 23, 24, 25

[70] Fanchao Qi, Yangyi Chen, Mukai Li, Yuan Yao, Zhiyuan Liu, and Maosong Sun. ONION: A simple and effective defense against textual backdoor attacks. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9558–9566, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.752. URL `https://aclanthology.org/2021.emnlp-main.752`. 22, 24, 25, 29, 34, 72, 77, 92

[71] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018. 22

[72] Y. Liu, Y. Xie, and A. Srivastava. Neural trojans. In *2017 IEEE 35th International Conference on Computer Design (ICCD)*, pages 45–48, Los Alamitos, CA, USA, 11 2017. IEEE Computer Society. doi: 10.1109/ICCD.2017.16. URL `https://doi.ieeecomputersociety.org/10.1109/ICCD.2017.16`. 23

[73] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *CoRR*, abs/1712.05526, 2017. URL `http://arxiv.org/abs/1712.05526`.

[74] Guowen Xu, Hongwei Li, Hao Ren, Jianfei Sun, Shengmin Xu, Jianting Ning, Haomiao Yang, Kan Yang, and Robert H Deng. Secure and verifiable inference in deep neural networks. In *Annual Computer Security Applications Conference*, pages 784–797, 2020.

[75] Guowen Xu, Hongwei Li, Yun Zhang, Shengmin Xu, Jianting Ning, and Robert Deng. Privacy-preserving federated deep learning with irregular users. *IEEE Transactions on Dependable and Secure Computing*, 2020. 23

[76] Siddhant Garg, Adarsh Kumar, Vibhor Goel, and Yingyu Liang. Can adversarial weight perturbations inject neural backdoors. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2029–2032, 2020. 23

[77] Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pretrained models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2793–2806, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.249. URL `https://aclanthology.org/2020.acl-main.249`. 23, 24, 31, 36, 72, 86

[78] Linyang Li, Demin Song, Xiaonan Li, Jiehang Zeng, Ruotian Ma, and Xipeng Qiu. Backdoor attacks on pre-trained models by layerwise weight poisoning. *arXiv preprint arXiv:2108.13888*, 2021. 24, 30

[79] Shangwei Guo, Chunlong Xie, Jiwei Li, Lingjuan Lyu, and Tianwei Zhang. Threats to pre-trained language models: Survey and taxonomy. *arXiv preprint arXiv:2202.06862*, 2022. 23

[80] HuggingFace. Huggingface. `https://huggingface.co/models`. Accessed: 2021-10-01. 24, 28

[81] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 27, 40

[82] 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. *arXiv preprint arXiv:1907.11692*, 2019. 27, 84

[83] Hakan Erdogan. Sequence labeling: Generative and discriminative approaches. In *Proc. 9th Int. Conf. Mach. Learn. Appl.*, pages 1–132, 2010. 30

[84] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`. 30, 32

[85] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL `https://aclanthology.org/D16-1264`. 31

[86] 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*, pages 19–27, 2015. 31

[87] Jesse Vig. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 37–42, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-3007. URL `https://www.aclweb.org/anthology/P19-3007`. 34, 40

[88] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, 2019. 34, 40

[89] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 38

[90] Zehao Dou, Wei Wei, and Xiaojun Wan. Improving word embeddings for antonym detection using thesauri and sentiwordnet. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 67–79. Springer, 2018. 38

[91] Kangjie Chen, Xiaoxuan Lou, Guowen Xu, Jiwei Li, and Tianwei Zhang. Clean-image backdoor: Attacking multi-label models with poisoned labels only. In *The Eleventh International Conference on Learning Representations*, 2022. 45, 73, 74, 76, 77, 79

[92] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, and Yanwen Guo. Multi-Label Image Recognition with Graph Convolutional Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 46, 48, 49, 51, 57

[93] Hao Guo, Kang Zheng, Xiaochuan Fan, Hongkai Yu, and Song Wang. Visual attention consistency under image transforms for multi-label image classification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 729–739, 2019. 46, 48

[94] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 46, 48

[95] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection. *arXiv preprint arXiv:2203.03605*, 2022. 46, 48

[96] Eneldo Loza Mencía and Johannes Fürnkranz. Efficient multilabel classification algorithms for large-scale problems in the legal domain. In *Semantic Processing of Legal Texts*, pages 192–215. Springer, 2010. 46, 48

[97] Sophie Burkhardt and Stefan Kramer. Online multi-label dependency topic models for text classification. *Machine Learning*, 107(5):859–886, 2018. 46, 48

[98] Shih-Han Chan, Yinpeng Dong, Jun Zhu, Xiaolu Zhang, and Jun Zhou. Baddet: Backdoor attacks on object detection. *arXiv preprint arXiv:2205.14497*, 2022. 46, 49

[99] Hua Ma, Yinshan Li, Yansong Gao, Alsharif Abuadbba, Zhi Zhang, Anmin Fu, Hyoungshick Kim, Said F Al-Sarawi, Nepal Surya, and Derek Abbott. Dangerous cloaking: Natural trigger based backdoor attacks on object detectors in the physical world. *arXiv preprint arXiv:2201.08619*, 2022. 46, 49

[100] Eva Gibaja and Sebastián Ventura. A tutorial on multilabel learning. *ACM Computing Surveys (CSUR)*, 47(3):1–38, 2015. 47

[101] Yunchao Wei, Wei Xia, Min Lin, Junshi Huang, Bingbing Ni, Jian Dong, Yao Zhao, and Shuicheng Yan. Hcp: A flexible cnn framework for multi-label image classification. *IEEE transactions on pattern analysis and machine intelligence*, 38(9):1901–1907, 2015. 49

[102] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2285–2294, 2016. 49, 51

[103] Vacit Oguz Yazici, Abel Gonzalez-Garcia, Arnau Ramisa, Bartlomiej Twardowski, and Joost van de Weijer. Orderless recurrent models for multi-label classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13440–13449, 2020. 49, 51

[104] Ya Wang, Dongliang He, Fu Li, Xiang Long, Zhichao Zhou, Jinwen Ma, and Shilei Wen. Multi-label classification with label graph superimposing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12265–12272, 2020. 49

[105] Shilong Liu, Lei Zhang, Xiao Yang, Hang Su, and Jun Zhu. Query2label: A simple transformer way to multi-label classification. *arXiv preprint arXiv:2107.10834*, 2021. 49

[106] Tal Ridnik, Gilad Sharir, Avi Ben-Cohen, Emanuel Ben-Baruch, and Asaf Noy. Ml-decoder: Scalable and versatile classification head, 2021. 49, 57

[107] Hossein Souri, Micah Goldblum, Liam Fowl, Rama Chellappa, and Tom Goldstein. Sleeper agent: Scalable hidden trigger backdoors for neural networks trained from scratch. *arXiv preprint arXiv:2106.08970*, 2021. 49

[108] Yuezun Li, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu. Invisible backdoor attack with sample-specific triggers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16463–16472, 2021. 49

[109] Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1505–1521, 2021. 49

[110] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *European Conference on Computer Vision*, pages 182–199. Springer, 2020. 49

[111] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *Advances in neural information processing systems*, 31, 2018. 49

[112] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. Composite backdoor attack for deep neural network by mixing existing benign features. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 113–131, 2020. 49

[113] Mengchen Zhao, Bo An, Wei Gao, and Teng Zhang. Efficient label contamination attacks against black-box learning models. In *IJCAI*, pages 3945–3951, 2017. 50

[114] Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on Machine Learning*, pages 8230–8241. PMLR, 2020.

[115] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 50

[116] Ding Wang, Shantanu Prabhat, and Nithya Sambasivan. Whose AI dream? in search of the aspiration in data annotation. In *CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2022. 51

[117] Market Analysis. Autonomous vehicle data annotation market analysis. `https://www.researchandmarkets.com/reports/4985697/autonomous-vehicle-data-annotation-market-analysis`, 2022. Accessed: 2022-08-07. 51

[118] Chen Edwin. 30% of google's emotions dataset is mislabeled. `https://www.surgehq.ai/blog/30-percent-of-googles-reddit-emotions-dataset-is-mislabeled`, 2022. Accessed: 2022-08-07. 51

[119] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 56

[120] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 56

[121] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 115–124, 2017. 68

[122] David D Lewis, Yiming Yang, Tony Russell-Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004. 68

[123] Tencent. Neuralclassifier: An open-source neural hierarchical multi-label text classification toolkit. `https://github.com/Tencent/NeuralNLP-NeuralClassifier`, 2019. Accessed: 2022-11-16. 68

[124] Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, and Yang Zhang. Badnl: Backdoor attacks against NLP models. In *ICML 2021 Workshop on Adversarial Machine Learning*, 2021. 72, 75, 78

[125] Yinghua Gao, Yiming Li, Linghui Zhu, Dongxian Wu, Yong Jiang, and Shu-Tao Xia. Not all samples are born equal: Towards effective clean-label backdoor attacks. *Pattern Recognition*, 139:109512, 2023. 72

[126] Amazon Mechanical Turk. https://www.mturk.com/. 72

[127] ByteBridge. https://bytebridge.org/. 72

[128] Rishi Dev Jha, Jonathan Hayase, and Sewoong Oh. Label poisoning is all you need. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=prftZp6mDH`. 73, 76, 77, 79, 86

[129] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. *ArXiv*, abs/2109.01652, 2021. URL `https://api.semanticscholar.org/CorpusID:237416585`. 75

[130] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. Instruction tuning for large language models: A survey, 2023. 75

[131] Tuan Anh Nguyen and Anh Tran. Input-aware dynamic backdoor attack. *Advances in Neural Information Processing Systems*, 33:3454–3464, 2020. 77

[132] Jiazhao Li, Yijin Yang, Zhuofeng Wu, V. G. Vinod Vydiswaran, and Chaowei Xiao. Chatgpt as an attack tool: Stealthy textual backdoor attack via blackbox generative model trigger, 2023. 78, 79, 80, 83, 87, 89

[133] Rafael Rivera Soto, Kailin Koch, Aleem Khan, Barry Chen, Marcus Bishop, and Nicholas Andrews. Few-shot detection of machine-generated text using style representations, 2024. 79

[134] Minshuo Chen, Song Mei, Jianqing Fan, and Mengdi Wang. An overview of diffusion models: Applications, guided generation, statistical rates and optimization. *arXiv preprint arXiv:2404.07771*, 2024. 79

[135] OpenAI. New ai classifier for indicating ai-written text. *OpenAI blog*, 2023. URL `https://openai.com/blog/new-ai-classifier-for-indicating-ai-written-text`. 79, 87

[136] Jonas Ricker, Simon Damm, Thorsten Holz, and Asja Fischer. Towards the detection of diffusion model deepfakes. *arXiv preprint arXiv:2210.14571*, 2022. 79

[137] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, 2013. 83

[138] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011. 83

[139] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, volume 28, pages 649–657, 2015. 83

[140] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. Supernaturalinstructions:generalization via declarative instructions on 1600+ tasks. In *EMNLP*, 2022. 84

[141] OpenAI, Josh Achiam, and et al. Gpt-4 technical report, 2023. 84

[142] Hugo Touvron, Louis Martin, Kevin Stone, and et. al. Llama 2: Open foundation and fine-tuned chat models, 2023. 84

[143] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.703. URL `http://dx.doi.org/10.18653/v1/2020.acl-main.703`. 84

[144] Maxim Kuznetsov Vladimir Vorobev. Chatgpt paraphrases dataset. 2023. 84

[145] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019. 84, 89

[146] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020. 85

[147] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018. 85

[148] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023. 85

[149] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://aclanthology.org/W04-1013`. 85

[150] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1875–1885, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1170. URL `https://aclanthology.org/N18-1170`. 86

[151] Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2525–2534. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/katharopoulos18a.html`. 88

[152] Zihao Zhu, Mingda Zhang, Shaokui Wei, Li Shen, Yanbo Fan, and Baoyuan Wu. Boosting backdoor attack with a learnable poisoning sample selection strategy, 2024. URL `https://openreview.net/forum?id=uDNP1q5aZq`. 88

[153] Alexander Wan, Eric Wallace, Sheng Shen, and Dan Klein. Poisoning language models during instruction tuning. In *International Conference on Machine Learning*, pages 35413–35425. PMLR, 2023. 88

[154] Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SkeHuCVFDr`. 89

[155] Chuanshuai Chen and Jiazhu Dai. Mitigating backdoor attacks in lstm-based text classification systems by backdoor keyword identification. *Neurocomputing*, 452:253–262, September 2021. ISSN 0925-2312. doi: 10.1016/j.neucom.2021.04.105. URL `http://dx.doi.org/10.1016/j.neucom.2021.04.105`. 92

[156] Ganqu Cui, Lifan Yuan, Bingxiang He, Yangyi Chen, Zhiyuan Liu, and Maosong Sun. A unified evaluation of textual backdoor learning: Frameworks and benchmarks. In *Proceedings of NeurIPS: Datasets and Benchmarks*, 2022. 92, 93

[157] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal. Strip: a defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19. ACM, December 2019. doi: 10.1145/3359789.3359790. URL `http://dx.doi.org/10.1145/3359789.3359790`. 92, 101

[158] Wenkai Yang, Yankai Lin, Peng Li, Jie Zhou, and Xu Sun. Rap: Robustness-aware perturbations for defending against backdoor attacks on nlp models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.659. URL `http://dx.doi.org/10.18653/v1/2021.emnlp-main.659`. 92

[159] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 1(4):7, 2009. 96

[160] 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*, pages 770–778, 2016. 96

[161] Sheng-Yen Chou, Pin-Yu Chen, and Tsung-Yi Ho. How to backdoor diffusion models? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4015–4024, 2023. 97

[162] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proc. of the NeurIPS*, pages 6626–6637, 2017. 100

[163] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *Proc. of the ICPR*, pages 2366–2369, 2010. 100

[164] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proc. of the ICML*, volume 139, pages 8748–8763, 2021. 100

[165] Kangjie Chen, Shangwei Guo, Tianwei Zhang, Shuxin Li, and Yang Liu. Temporal watermarks for deep reinforcement learning models. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 314–322, 2021. 109

[166] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 110, 113, 122, 125

[167] Ruimin Shen, Yan Zheng, Jianye Hao, Zhaopeng Meng, Yingfeng Chen, Changjie Fan, and Yang Liu. Generating behavior-diverse game ais with evolutionary multi-objective deep reinforcement learning.

[168] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning.

In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 772–784. IEEE, 2019. 110

[169] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017. 110

[170] Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Ruoxi Jia, Bo Li, and Dawn Song. REFIT: A unified watermark removal framework for deep learning systems with limited data. *arXiv preprint arXiv:1911.07205*, 2019. 111, 115

[171] William Aiken, Hyoungshick Kim, and Simon Woo. Neural network laundering: Removing black-box backdoor watermarks from deep neural networks. *arXiv preprint arXiv:2004.11368*, 2020.

[172] Shangwei Guo, Tianwei Zhang, Han Qiu, Yi Zeng, Tao Xiang, and Yang Liu. The hidden vulnerability of watermarking for deep neural networks. *arXiv preprint arXiv:2009.08697*, 2020. 111, 115

[173] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pages 363–372. 2006. 113

[174] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation*, pages 3357–3364, 2017. 113

[175] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 113

[176] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 114, 122, 125

[177] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016. 114

[178] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 115

[179] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016. 115

[180] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in Neural Information Processing Systems*, pages 5279–5288, 2017. 115

[181] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988. 123

[182] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. 124

[183] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. 125, 126

[184] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. 126

[185] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017. 127

[186] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 129

[187] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015. 129

[188] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 130

[189] Anthony R Cassandra. A survey of pomdp applications. In *Working notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*, volume 1724, 1998. 130

[190] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 136