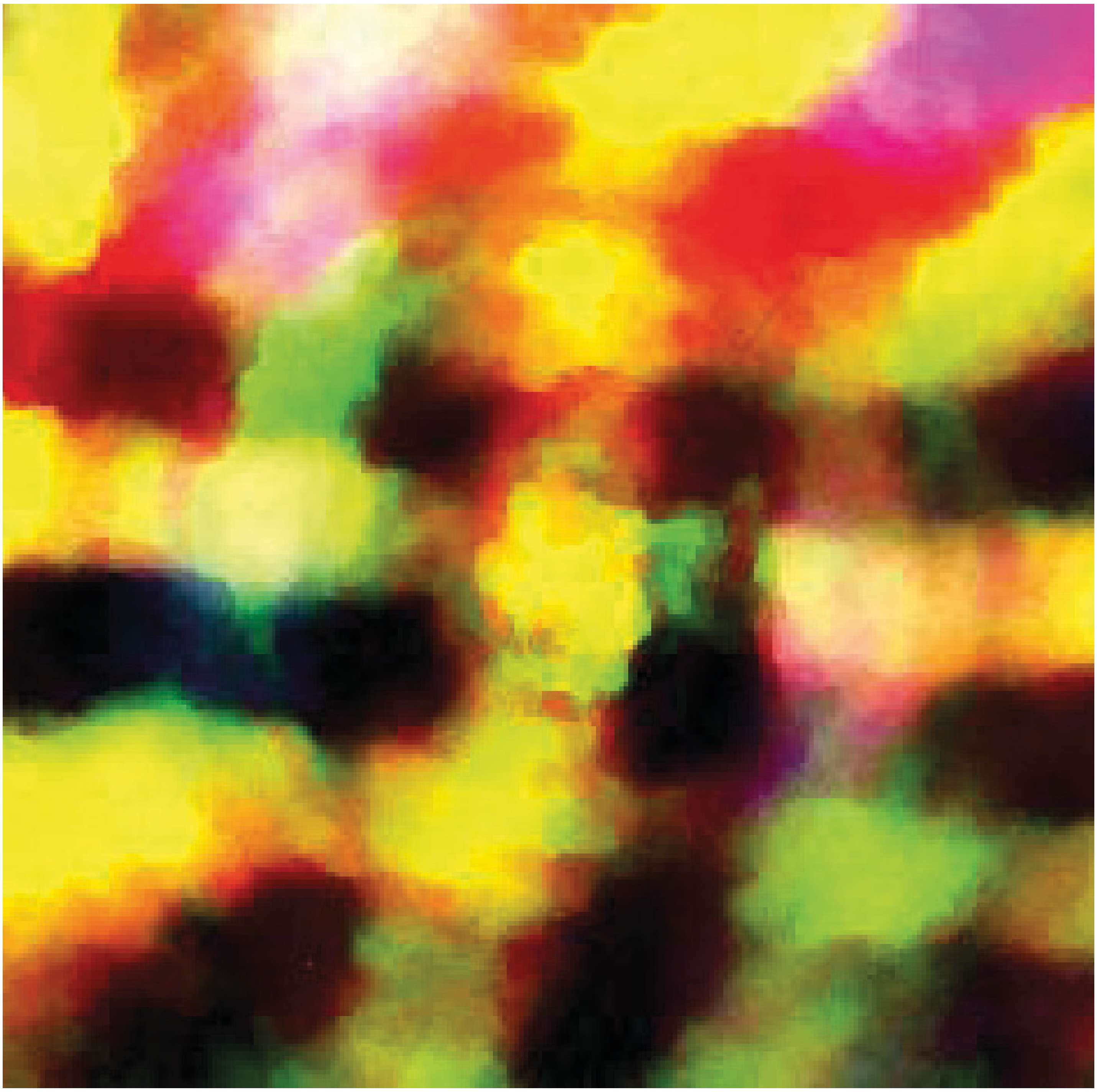
Attacking and Deceiving Military AI Systems

FARZAD KAMRANI, LINUS KANESTAD,
LINUS J. LUOTSINEN, BJÖRN PELZER,
JOHAN SABEL, VIKTOR SANDSTRÖM,
AGNES TEGEN



FOI-R--5396--SE
ISSN 1650-1942 March 2023

FOI-R--5396--SE

Farzad Kamrani, Linus Kanestad,
Linus J. Luotsinen, Björn Pelzer,
Johan Sabel, Viktor Sandström,
Agnes Tegen

Attacking and Deceiving Military AI Systems

Titel	Angrepp och vilseledning av militära AI-system
Title	Attacking and Deceiving Military AI Systems
Report no	FOI-R--5396--SE
Month	March
Year	2023
Pages	46
ISSN	ISSN-1650-1942
Customer	Swedish Armed Forces
FOI Research area	C3 and Human Factors
Armed Forces R&T area	Command and Control
Project no	E38519
Approved by	Linda Sjödin
Division	Cyber Defence and C2 Technology
Export control	The content has been reviewed and does not contain information which is subject to Swedish export control.

Bild/Cover: adversarial patch, generated using `https://github.com/andrewpatrickdu/adversarial-yolov3-cowc`

Summary

This report investigates adversarial machine learning (AML), the research into methods of exploiting weaknesses in AI systems based on machine learning (ML). In recent years, machine learning, especially deep learning (DL), has allowed rapid progress in diverse fields like image classification, natural language processing and autonomous agents. As such DL is also of interest in military contexts. Yet, alongside the progress there has been a rising interest in AML methods, with new attack variations being published constantly. Practically all DL-systems are susceptible in some way, whether it is to confuse them, to avoid being detected by them, or to extract secret information they may hold. From a military perspective it is important to be aware of the possibility of such exploits, both against the own AI systems and against those used by an adversary.
The report provides an overview of AML research, and then showcases a selection of attack methods against different types of AI systems:

- *poisoning* of image classification systems, enabling military vehicles to avoid detection;
- *extraction* attacks that can retrieve secret information from large generative models;
- *adversarial policy* attacks where an adversary behaves in a manner that confound autonomous agents.

Each case describes and discusses the attacks and evaluates implementations. The focus of this report is on the attacks. While defence against AML methods is discussed briefly where applicable, a more in-depth study of AML defence is the subject of a follow-up report.

Keywords: artificial intelligence, machine learning, deep learning, deep neural networks, deception, cyber attacks, attack vectors, vulnerabilities, adversarial examples, data poisoning, data extraction, adversarial policy

Sammanfattning

Denna rapport studerar AML (eng. *adversarial machine learning*, fientlig maskininlärning), forskningen om metoder som exploaterar svagheter i AI-system som använder sig av maskininlärning (ML). Under de senaste åren har maskininlärning, och särskilt djupinlärning (DL), lett till snabba framsteg i många olika områden såsom bildklassificering, NLP (eng. *natural language processing*, språkteknologi) och autonoma agenter. DL är därför av stort intresse inom militära sammanhang. Men parallellt med framstegen ökar också forskningen inom AML, och nya attackvarianter publiceras nästan dagligen. Praktiskt taget alla DL-system är sårbara i någon form, vare sig att de kan bli förvirrade, att en motståndare kan undvika att detekteras, eller att hemlig information kan extraheras ur systemen. Från ett försvarsperspektiv är det viktigt att vara medveten om möjligheten att sådana angrepp kan genomföras, både mot egna AI-system och mot de av en motståndare.
Rapporten ger en översikt över forskningen inom AML. Sedan presenteras tre fallstudier om angreppsvarianter mot olika typer av AI-system:

- *förgiftning* (eng. *poisoning*) av bildklassificerare, så att militära fordon undviker att bli upptäckta;
- *extraktion* (eng. *extraction*) av hemlig information ur stora generativa modeller;
- *fientlig policy* (eng. *adversarial policy*): attacker där motståndaren beter sig på ett sätt som vilseleder autonoma agenter.

Varje fallstudie beskriver och diskuterar attackerna och evaluerar implementeringar.
Rapporten fokuserar på angreppen. Försvar mot AML diskuteras kort där det är lämpligt, men en mer djupgående studie av AML-försvar är tema för en kommande rapport.

Nyckelord: artificiell intelligens, maskininlärning, djupinlärning, djupa neuronnät, vilseledning, cyberangrepp, attackvektorer, sårbarheter, manipulation av indata, dataförgiftning, dataextraktion, fientlig policy

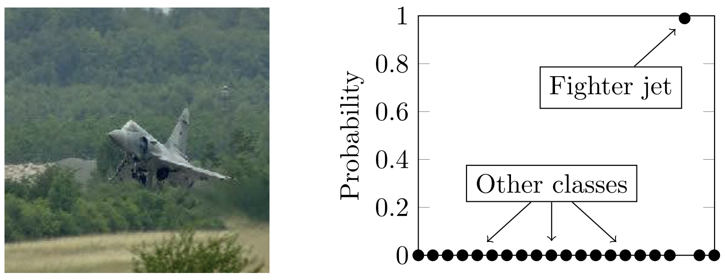
Contents

1 Introduction

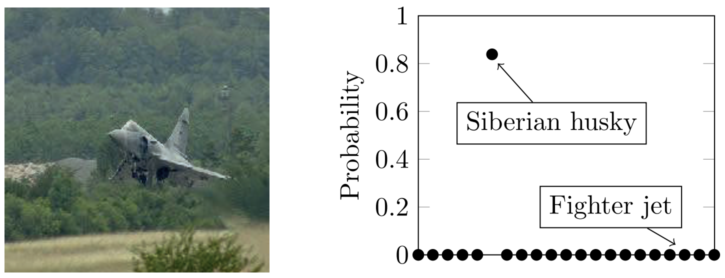
The advent of deep learning (DL) has brought the performance and capability of intelligent computer software into new levels of performance. Embedding DL-based software in military command, control, communications, computers, intelligence, surveillance and reconnaissance (C4ISR) systems has the potential to revolutionize the ability to create accurate and timely common operational pictures (COPs) such that military decision making processes can be performed faster and with greater precision than ever before. In the long, term DL may also be used to create military plans in complex warfare environments that stretch far beyond what humans are capable of.

However, DL-based software, implemented by deep neural networks (DNNs), is vulnerable to a breed of threats or cyber attacks. These are studied and developed in the adversarial machine learning (AML) research field. The attacks can potentially be used to deceive decision makers, reduce system performance, lower end user trust and even extract (i.e., reverse engineer) sensitive military data from the system. Figure 1.1 illustrates an example of a typical AML attack where the target is a DNN used to classify image contents. In this case, the DNN is able to correctly recognize that the original image in Figure 1.1a contains a fighter jet with near perfect certainty. The malicious image in Figure 1.1b, which was created by applying AML techniques on the original image, is able to fool the same DNN into classifying the input as a Siberian husky instead of a fighter jet. In this case, the attack is effective despite being imperceptible to the human eye.

To the best of our knowledge AML has not yet been used by adversaries or advanced persistent threat (APT) actors to target and attack DL-based software embedded in real-world military systems. However, research teams and security experts continuously demonstrate that attacks are possible againstn a wide range of applications that rely on DL to achieve cutting edge performance [1]. For instance, carefully replacing words in sentences can cause language models to misclassify sentiments [2]. Traffic sign and lane detection systems used by autonomous cars can be attacked by placing stickers on the signs and the road respectively [3, 4]. Transcription services can be misled by injecting carefully designed noise forcing the system to convert speech into arbitrary text [5, 6]. As such, assuming that DL-based software will be ubiquitously used in future C4ISR support systems, it is expected that adversaries and APTs will eventually exploit these vulnerabilities to deceive, deny access or gather intelligence.



(a) Benign input and DNN classification output



(b) Malicious input and DNN classification output

Figure 1.1 – An example attack using AML. In this case the target is an image classification system represented by a DNN. Figure 1.1a shows that the DNN is able to correctly classify the benign (non-manipulated) input as a fighter jet with near perfect certainty. Figure 1.1b shows a manipulated image created using AML techniques. The manipulated image successfully fools the DNN to classify the input as a Siberian husky instead of a fighter jet.

1.1 Objective and Scope

The objectives of this report are: (1) to present an overview of the attack vectors that have been identified in the AML research field to date, (2) to empirically estimate the efficacy of a subset of these attacks in a military context, and finally (3) to provide insights and discuss to what degree AML is a realistic and serious threat in real-world military applications of DL.

Although AML is applicable to any ML-based system and algorithm, this report focuses on ML systems that are based on DL. Furthermore, this report will focus on attacks. Defence mechanisms that have been proposed and developed in the AML research field will be covered in future works. Finally, we limit the scope to applications of DL that are relevant in the context of command and control (C2), intelligence, surveillance and reconnaissance.

1.2 Target Readership

The target readership of this report is personnel that operate, acquire or develop military systems where AI, ML and DL technologies are used by or embedded in the systems.

1.3 Reading Instructions

This report assumes that the reader has basic knowledge about ML and DL concepts such as supervised learning, reinforcement learning, loss functions, gradient descent and backpropagation. Readers lacking such knowledge are encouraged to read chapter 2 in the FOI-report FOI-R–4849–SE [7] prior to proceeding with this report.

1.4 Outline

Chapter 2 provides an introduction to AML and presents a taxonomy that is used to categorize and compare attacks in this report. Chapter 3 presents three case studies of known attack methods that may become relevant from a military perspective. The methods are implemented and evaluated. Chapter 4 concludes the report with a discussion on the real-world applicability of AML, including in in the military domain.

2 Adversarial Machine Learning

AML is the combination of cyber security and ML [8] (Figure 2.1). The aim of AML is to design secure ML systems that can resist attacks performed by adversaries and, naturally, also to study and develop attack capabilities, limitations and consequences [9]. This chapter focuses primarily on the capabilities and limitations of attacks that have been proposed in the AML literature. Potential consequences of attacks on military systems are presented in Chapter 4.

2.1 AML Statistics and Trends

The AML research field has been active since at least 2004 [10, 11]. It is however not until recently that the field has gained significant attention. Figure 2.2 shows that the interest and the number of research contributions in this field started to grow in 2017 according to data extracted from Google Trends (Figure 2.2a) and the Arxiv database (Figure 2.2b) respectively. It is worth noting that more than 1,200 articles were submitted to the Arxiv database in 2021 and 2022.

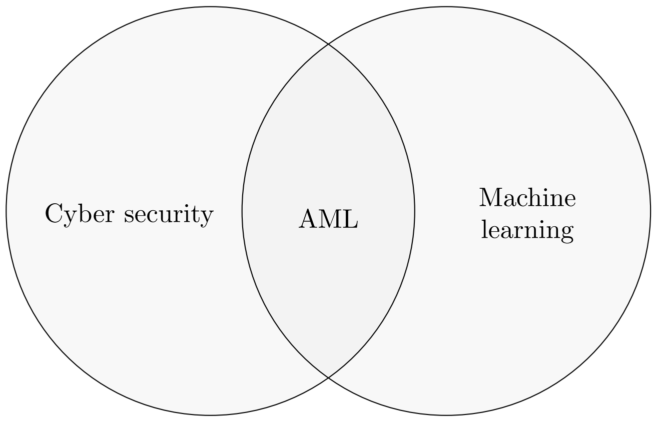
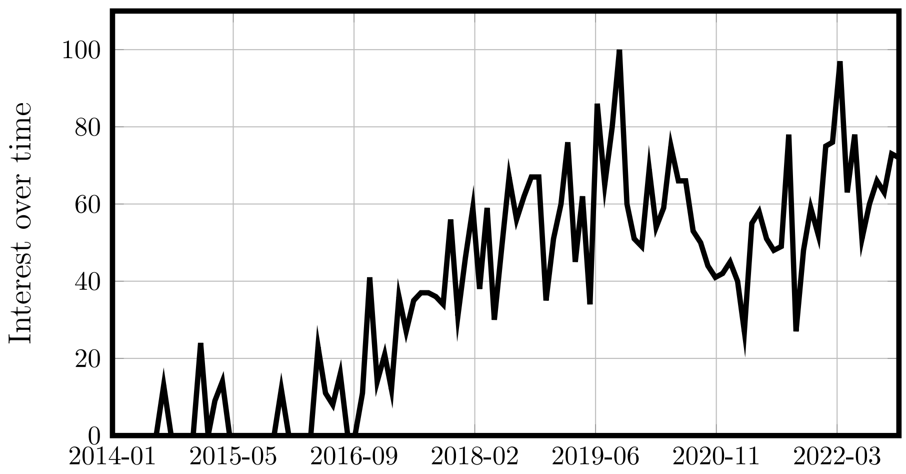
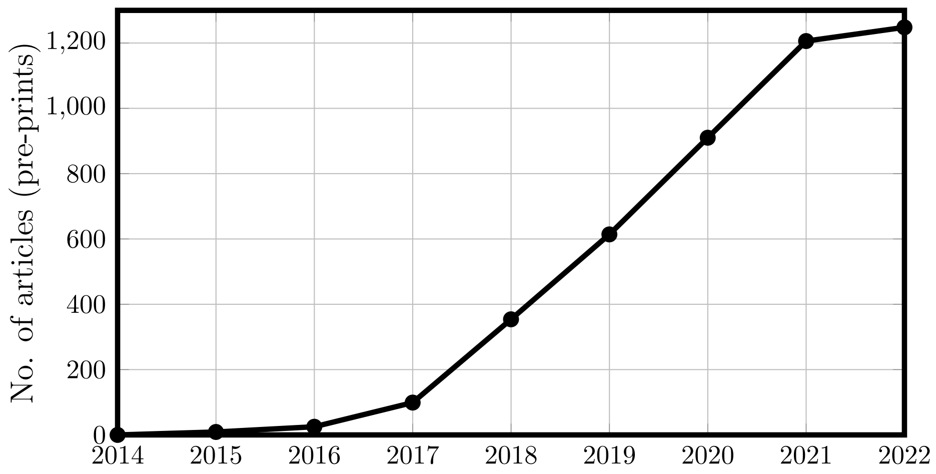


Figure 2.1 – AML is a research field that lies in the intersection of cyber security and machine learning. The main focus of the field is to ensure that ML-systems are secure and robust when deployed in environments where adversarial actions are expected.



(a) Interest over time for AML according to Google Trends. The term "adversarial machine learning" was used to acquire data based on web searches in the interval 2014-01-01 to 2022-12-31. Note that interest over time is a relative metric within the chosen interval. A value of 100 represents the peak of popularity and 50 means the topic is half as popular within the selected interval.



(b) Number of AML-related pre-print articles submitted to arxiv.org for each year in the interval 2014 to 2022. Article selection was performed by searching for abstracts that included at least one of the terms "adversarial machine learning", "adversarial example" or "adversarial attack". Note that no articles was submitted in 2014 whereas more than 1,200 articles were submitted in 2021 and 2022. In total the database contains 4,465 articles on the topic in this interval.

Figure 2.2 – Data acquired from Google Trends (Figure 2.2a) and arxiv.org (Figure 2.2b) reveals that the interest and scientific contributions in the AML-field started to increase in 2017.

2.2 Taxonomy of AML Attacks

The taxonomy presented in Figure 2.3 is used in this report to categorize and compare existing as well as future AML attacks. The taxonomy, which is adapted from previously developed taxonomies [12, 9, 11, 13, 8, 14, 15], consists of the following categories: (1) knowledge, (2) domain, (3) specificity, (4) type, (5) scope, and (6) perception. Each category is further separated into two or more subclasses.

2.2.1 Knowledge

The category *knowledge* refers to how much knowledge the attacker has of the ML system they are attacking. There exist three subclasses within this category, white-box attacks, grey-box attacks and black-box attacks.

White-box

In *white-box* attacks the adversary has full knowledge of the target ML system, including, e.g., training and testing data as well as parameters of the model. White-box attacks are therefore the easiest attacks to pursue for an attacker, but the most challenging case when defending against attacks. They are often used as worst-case scenarios during security analysis [16]. Athalye et al. claim that many defences are not as robust as they might appear and can provide a false sense of security [17]. They studied white-box defences presented in papers accepted at the International Conference on Learning Representations in 2018. They found that 7 out of 9 defences relied on a certain type of gradient masking called obfuscated gradients [18]. The authors focus on the defences with obfuscated gradients and managed to successfully circumvent 6 completely and one partially.

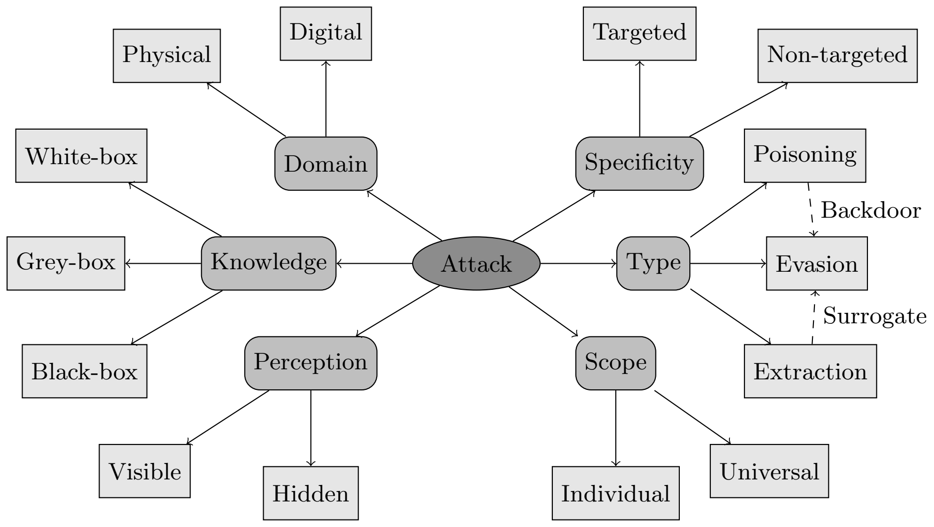


Figure 2.3 – Taxonomy for AML attacks from an adversary's perspective. The taxonomy consists of six classes (i.e., knowledge, domain, specificity, type, scope and perception). Each class in turn consists of two or more subclasses.

Grey-box

In the case of *grey-box* attacks the adversary has limited knowledge about the ML system [16]. For instance, they might have knowledge only about feature representation and optimization algorithms. The attacker can create a surrogate model based on the available knowledge and then transfer adversarial examples to the target ML model. The main idea is that an adversarial example that evades one model will evade other models that are similar.

Black-box

If the adversary does not have any knowledge of the model or its attributes, the attacks are defined as *black-box*. Black-box attacks are the most difficult type of attacks for the adversary to pursue. In this case, the attacker can only get information about the ML model by querying the system for labels or confidence scores. Papernot et al. constructed an attack based only on knowledge about labels given to the target ML algorithm for chosen inputs [18]. They showed that their attack was successful on multiple real-world scenarios, including evading defence strategies, which have previously been found to make adversarial example crafting more difficult.

2.2.2 Domain

The *domain* category describes whether the attack is in the physical or digital domain. More specifically, this category distinguishes whether the adversary has altered information in the physical world or digital data to reach their desired outcome.

Physical

While attacks in the digital domain might be more common and well-known, there are many application areas where possible attacks within the *physical* domain need to be considered. Attacks in the physical domain mean that the adversary alters elements in the physical world, typically to make the machine learning algorithm classify the target incorrectly. Athalye et al. proposed an algorithm for constructing adversarial examples over a selected distribution of transformations [19]. Based on the examples, they then created physical 2D and 3D objects which were adversarial over a large and realistic distribution of viewpoints.

Digital

The *digital* domain is where most attacks are carried out. There exists a variety of attacks within this category. For instance, in an image classification problem an input image might be manipulated to change the outcome from the classification. Figure 1.1 displays an example of this where the attack is hidden. Kurakin et al. discuss adversarial training, where the aim is to make a model more robust by training it on adversarial examples [20]. They address the risk of the adversarially trained model to perform better on adversarial examples than on clean examples, an effect called label leaking. In the case of label leaking, the model has learnt the regularities in the construction process of the adversarial examples.

2.2.3 Specificity

Specificity refers to how particular the attack is intended to be, i.e., does the adversary target specific classes in the output or is any outcome accepted as long as it is a false negative. The specificity of an attack is not binary but rather a continuous spectrum [12]. The two subclasses presented below, targeted and non-targeted attacks can be seen as the two extremes of the spectrum.

Targeted

At one end of the specificity spectrum are the *targeted* attacks. A targeted attack is focused on a specific point or a small set of points [12]. Carlini et al. study a defence strategy frequently used against adversarial attacks called defensive distillation [21]. They emphasize the importance of analysing how a defence might be attacked, not only against currently existing attacks, but also against future attacks. If an attack fails, it is relevant to understand why it failed when constructing defences.

Non-targeted

Opposite on the spectrum to the targeted attacks are the *non-targeted* attacks. They do not have a specific target for the attack, but have a more flexible goal where the adversary tries to find any opportunity to exploit and the aim is to misclassify as many samples as possible [12]. Liu et al. study transferability of adversarial examples in both targeted and non-targeted attacks [22]. Transferability is the possibility and success rate of transferring adversarial examples adapted for one model to another. They found that transferability was prominent for non-targeted attacks, even if the model and dataset were large.

2.2.4 Type

The *type* of attack describes what the aim of the attack is and the form it takes. The three subclasses of type are poisoning, evasion and extraction.

Poisoning

In *poisoning* attacks, the training data is manipulated by the adversary. For instance, data might be labelled as harmless when it actually is malicious through adversarial contamination. In Section 3.1, experiments with poisoning attacks within a military application are carried out. The experiments train an ML algorithm to recognize vehicles in images. The experiments focus on the ability of the ML algorithm to distinguish between military tanks and cars. Manipulated images are included in the training set, to perturb the classification, and the effect on the results are studied.

Evasion

Evasion is the most common type of attack. It does not involve modifying the training data, as with poisoning, but rather modifying test data (i.e., data used on the trained model). The aim of the attack is to evade detection by any potential defence. Elsayed et al. create adversarial examples that not only manage to evade multiple ML models, but also humans [23]. The authors point out that this is in contrast to the widely made assumption that while ML classifiers can be fooled by adversarial examples, humans can not. Evasion attacks are also applicable to models trained via reinforcement learning rather than examples; Section 3.3 investigates several variations of *adversarial policy* attacks against such models. Nonetheless, there is a major difference between attacks on reinforcement learning models compared to other machine learning methods as reinforcement learning models are trained to solve sequential decision-making problems in contrast to most other machine learning methods that are trained to solve single-step prediction problems [16].

Extraction

Extraction attacks do not modify data, unlike poisoning and evasion. Instead they aim at extracting information from a trained ML model. The information can then be used to reconstruct the ML model by the adversary. In other cases, the adversary might simply want access to the information itself, which can be confidential or sensitive. In Section 3.2, extraction is studied through experiments on language models. The aim is to extract information regarding the data which the model is trained on. The section includes a defensive experiment where unique patches are added to language data, which makes the model less susceptible to the attack by making it more difficult for the model to memorize data.

2.2.5 Scope

An attack can be constructed so that it alters a specific input or alters input in general. *Scope* describes this characteristic of an attack, which can be universal or individual.

Individual

Individual attacks are the most common attack. They generate an individual perturbation used to manipulate each individual example [24].

Universal

Universal attacks create one perturbation, which can then be used for all input data [24]. Universal attacks are easier to deploy in the real world, compared to individual attacks, as the perturbation does not need to change, even if the input changes. Madry et al. [25] utilize a universal adversary in their experiments and present evidence that DNN can be made resistant to adversarial attacks. Their focus is on certain datasets and they claim that further work will lead to adversarially robust networks. Still, it is important to remember the back-and-forth between attacks and defences, as soon as a new defence has been introduced, new attacks start to be developed.

2.2.6 Perception

Perception describes whether an attack is detectable or not for a human, by only looking at the input data for instance.

Hidden

A hidden attack is, as the name suggests, not visible to a human. An example of a hidden attack is the work by Elsayed et al. [23], where they manage to trick humans with manipulated input. Figure 1.1 contains another example where the attack is hidden.

Visible

Visible attacks are visible to a human eye, yet manage to fool the ML system if successful. This could be for instance a portion of an image that is modified, such as a trigger patch (described further in Section 3.1).

3 Case Studies

This chapter presents three case studies, which explore different types of attacks against ML-based systems. In each case one type of attack method is chosen from the AML-literature and implemented or tested from a military perspective. The efficacy of the attack is evaluated, followed by a discussion on practical considerations. The three case studies were selected due to their potential relevance for the military domain, to cover a broad range of attacks, and to illustrate a variety of ML applications and methods.

Chapter 1 opened with the example of deceiving a DNN into misinterpreting an image of a fighter jet as a dog. While hiding military equipment in plain sight has obvious appeal, the introductory example is highly idealized. A practical application faces a hurdle in that the attack is limited to the digital domain: The manipulation is performed on the digital image itself, that is, at a stage after the fighter jet was photographed. If the image was created by the adversary (e.g., the jet was filmed by a surveillance camera), manipulating the image would require deep access into the enemy systems. This is unlikely to be available (and if it is, simpler and more robust attacks become feasible, such as eliminating the image or preventing its recording). Furthermore, while black-box knowledge about the targeted DNN can be sufficient to compute the required image modifications (e.g., observation of classification label results [18]), in practice even this knowledge cannot be expected.

The first case study in Section 3.1 therefore investigates *data poisoning.* The objective of this attack is the same as in the introductory example: enable military vehicles (in this case tanks) to evade detection by deceiving the enemy DNN into misclassification of the vehicles. Still, while the methods are similar as well, the poisoning attack addresses the practical shortcomings of the introductory example.

Section 3.2 expands the scope to attacks on language models via *data extraction.* Language models are very large DNNs trained on extensive text corpora (typically many billions of words), enabling in some sense an "understanding" of (written) language. They have caused a paradigm change in natural language processing, setting new benchmarks in numerous tasks [26], and garnering much media attention for their ability to generate text [27]. Indeed, progress even during the compilation of this report has been remarkable, for example with the presentation of the *ChatGPT* system[1]. Language models are continuously closing in on human levels of natural language processing, and their potential effects and consequences for virtually all aspects of society, including military applications, are difficult to predict at this time. Besides opportunities, they also bring risks, for example in that they may expose sensitive information to an adversary. The case study in Section 3.2 investigates the feasibility of this form of adversarial extraction attack.

Section 3.3 studies attacks on models trained via reinforcement learning. Such models are typically used for autonomous agents in unmanned vehicles, robots, games and similar. They are not trained in a supervised manner on a fixed set of examples. Instead, the agent evaluates its situation with a reward function and chooses a course of action that maximizes the reward. While this mode of operation provides agents with flexibility and resilience for dealing with the real world, they are nevertheless susceptible to attacks and deception, as this case study will demonstrate on a variety of systems based on reinforcement

[1]`https://openai.com/blog/chatgpt/`

learning.

3.1 Data Poisoning

This case study examined a data poisoning technique that is used to deceive DL-based computer vision models by making them misclassify certain images. Specifically, models used to classify images as either “tank” or “car” were poisoned (see Section 2.2) during the training process by manipulating their training data. The goal was to trick the models into classifying images of military tanks as “car”, but only for tank images that had been injected with a trigger. In this context, *trigger* refers to a small image patch that has been pasted on an image (the patch contains a specific trigger pattern).

The poisoning attacks described in this case study are *digital attacks* in the sense that the trigger is added during post-processing of the images. However, in theory, it would be possible to perform similar *physical attacks* by attaching a physical trigger to real-world objects before any images are captured. Hence, one could imagine a tank bypassing a DL-based military vehicle detection system when covered with just a small physical trigger patch.

Furthermore, the poisoning attack itself is executed before the targeted DL-system is deployed, by manipulating the model during its creation and training. The subsequent actual evasion requires no complex additional manipulation of the deployed detector, nor any access into its operation.

3.1.1 Hidden Trigger Backdoor Attacks

A naive approach to poisoning a training dataset of images is to paste a trigger patch on some of the images in the dataset and change the associated class label [28]. For instance, if the training dataset contains images of military tanks and civilian passenger cars, an adversary could poison the dataset by pasting the trigger on some of the tank images and label those images as “car”. If a victim then trains a DL-based binary classification model on the poisoned dataset, the trained model will tend to misclassify unseen test images of tanks as “car” when the trigger pattern is present in the tank images. The reason for this is that the model has learnt to associate the characteristic trigger pattern with the class label “car”. This data poisoning approach is illustrated in Figure 3.1.

One limitation of the attack described above is that the trigger is clearly visible, even for a human, when briefly inspecting the training images. Furthermore, the images that contain the trigger are mislabelled. This makes it less likely that a victim would actually train a model on such images.

Hidden trigger backdoor attacks [29] are a type of data poisoning attack where the DL-based classification model is only trained on images that do not contain a visible trigger. All training images are also correctly labelled. Although the trigger pattern is not revealed during training, it can still be pasted onto images at test time, thus causing the trained model to misclassify the images.

Figure 3.2 illustrates the concept behind the hidden trigger backdoor attack proposed by Saha, Subramanya, and Pirsiavash [29] (the figure has been adapted for the two classes used in this case study). The basic poison generation idea is, through optimization, to find a poisoned car image car_{poisoned} (i.e., an altered car image) that is close to a corresponding original car image car_{original} in pixel space, but also close to a patched tank image $tank_{\text{patched}}$ in feature space. In other words, the goal is to generate a poisoned image that visually looks like the original car image, but has similar features as the tank

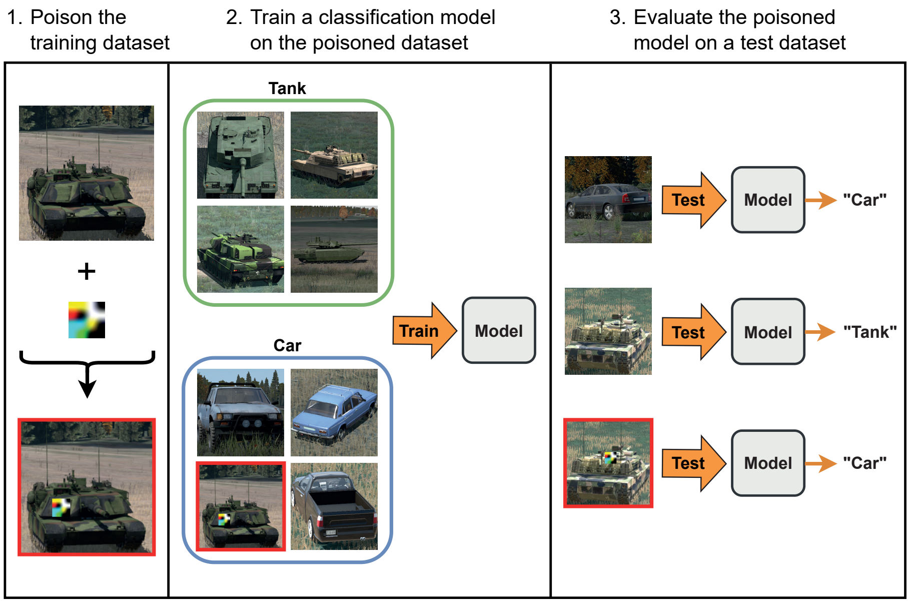

1. Poison the training dataset
2. Train a classification model on the poisoned dataset
3. Evaluate the poisoned model on a test dataset
+
Tank
Car
Train
Model
Test
Model
"Car"
Test
Model
"Tank"
Test
Model
"Car"

Figure 3.1 – Data poisoning attack using a visible trigger and mislabelled training images. In this figure, all patched tank images have been outlined in red for clarity.

image with a trigger patch. In this case, *features* refer to the intermediate features $f(\cdot)$ of some pre-trained DL-based classification model, such as AlexNet or ResNet. Note that this pre-trained model is only used as a reference during poison generation, for the purpose of obtaining the feature representations of the patched tank image. This model is *not* being trained further; its parameters are frozen.

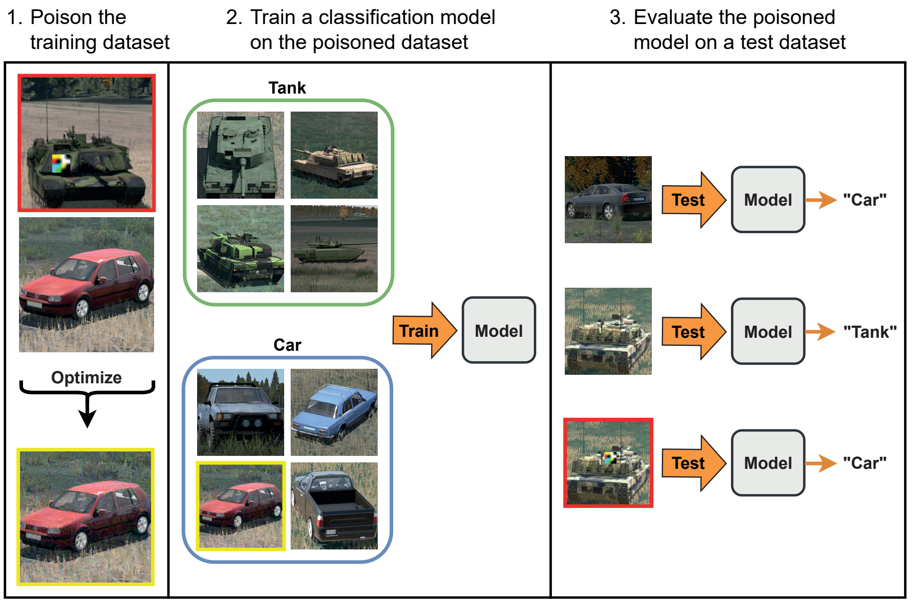

1. Poison the training dataset
2. Train a classification model on the poisoned dataset
3. Evaluate the poisoned model on a test dataset
Optimize
Tank
Car
Train
Model
Test
Model
"Car"
Test
Model
"Tank"
Test
Model
"Car"

Figure 3.2 – Data poisoning attack using only correctly labelled training images and a hidden trigger that is not revealed until test time. In this figure, all patched tank images have been outlined in red, while all poisoned car images have been outlined in yellow.

Formally, the poison generation algorithm [29] alters the pixel values of the poisoned car image to minimize the distance $\|f(car_{\text{poisoned}}) - f(tank_{\text{patched}})\|_2^2$ between the poisoned car image features $f(car_{\text{poisoned}})$ and the patched tank image features $f(tank_{\text{patched}})$. This is done subject to the constraint that $\|car_{\text{poisoned}} - car_{\text{original}}\|_\infty < \epsilon$, where $\epsilon \approx 16$ when image pixel values range between $0 - 255$. The constraint ensures that *each* pixel value of the poisoned car image remains relatively close to the corresponding original value, i.e., it will still look like the poisoned image contains a car after the pixel values have been altered. Note that this explanation is slightly simplified and omits a few algorithmic details necessary to generalize the data poisoning attack [29].

The final poisoned training dataset is built by adding some of the poisoned car images to a dataset already containing clean images of tanks and cars that have not been altered in any way (Figure 3.2). If a victim fine-tunes a pre-trained DL-based classification model on the poisoned dataset, the model will learn to associate the poisoned car images with the class label “car”. However, since the poisoned car images have feature representations that are similar to those obtained from patched tank images, the model will also associate patched tank images with the class label “car” even though the training dataset does not contain any patched tank images. Hence, the trigger pattern is not revealed during fine-tuning.

If the data poisoning attack is successful, the fine-tuned model will tend to classify clean car images as “car”, clean tank images as “tank”, and patched tank images as “car” (Figure 3.2). This model behaviour makes the victim believe that the model works as intended; at least until the adversary starts using the trigger pattern.

3.1.2 Experimental Setup

The description of the experimental setup used in this case study can be divided into: (1) the process behind generating clean images of cars and tanks; (2) the process behind generating poisoned images of cars; and, (3) the approach for fine-tuning classifiers on poisoned datasets and testing their performance.

Generating clean images of cars and military tanks

It is difficult to find sufficiently large datasets containing images of military tanks. Therefore, a dataset containing clean images of tanks and cars was generated using the software Virtual Battle Space 3 (VBS3) from Bohemia Interactive Simulations. The generated dataset contains 10 tank models and 10 car models (example images are shown in Figure 3.3). 1,000 images of each vehicle model were generated, i.e., the dataset contains 20,000 images in total. The images were captured at different locations in the simulator using different camera angles, and each car model was generated in different colours while each tank model was generated using different camouflage paint schemes.

Generating poisoned images of cars

Two DL-based classification models were used to generate poisoned car images: AlexNet [30] and ResNet-18 [31]. Both models were available with pre-trained weights on the ImageNet dataset [32] that contains 1,000 object classes and approximately one million images. In this case study, each pre-trained model was used on 1,000 clean car images and 1,000 clean tank images (different images for each model) to generate 1,000 poisoned car images. In other words, 2,000 car images and 2,000 tank images were collected from our dataset in order to generate 2,000 poisoned car images. This was achieved by pasting

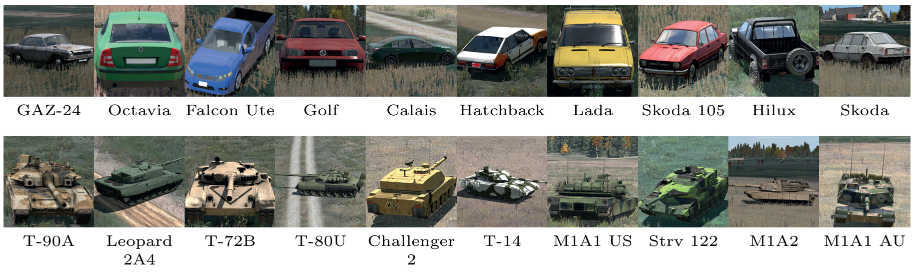
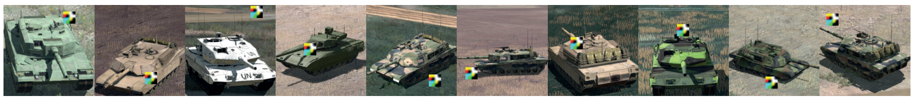
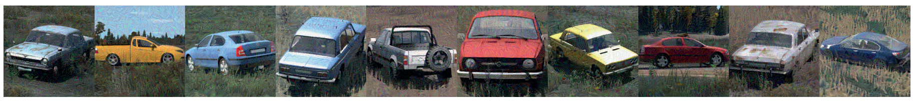


Figure 3.3 – Images of 10 car models and 10 tank models available in VBS3.

a trigger patch on a random location in each tank image and altering the pixel values of the car images following the optimization scheme proposed by Saha, Subramanya, and Pirsiavash [29], which is described in Section 3.1.1. All images had a resolution of 224×244 pixels, while the trigger patch had a resolution of 30×30 pixels (i.e., it only covered 1.8 % of each tank image). Some of the patched tank images and poisoned car images are shown in Figure 3.4.


(a) Patched tank images used when generating poisoned car images.


(b) Poisoned car images generated using AlexNet.

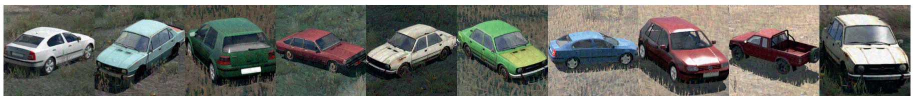

(c) Poisoned car images generated using ResNet.

Figure 3.4 – Patched military tank images, and poisoned car images generated using AlexNet and ResNet.

Fine-tuning classification models on poisoned training datasets and testing their performance

After the poisoned car images had been generated, pre-trained classification models were fine-tuned on training datasets containing clean car images (labelled as "car"), clean tank images (labelled as "tank"), and poisoned car images (labelled as "car"). During fine-tuning, the models were trained to classify the images as either "car" or "tank", guided by the labels to gradually improve performance. The fine-tuned models were then tested on clean car images, clean tank images, and patched tank images covered with the same trigger pattern used during poison generation. Note that different images were used for poison generation, fine-tuning, and testing.

To be more specific, two classification models were fine-tuned and tested: AlexNet and ResNet-18 (both pre-trained on ImageNet). The AlexNet and ResNet models used during poison generation were not reused during fine-tuning and testing (new instances of the models were always used). By using AlexNet and ResNet models it was possible to measure the test performance of AlexNet after being fine-tuned on poisoned images generated using ResNet, and ResNet after being fine-tuned on poisoned images generated using AlexNet. Saha, Subramanya, and Pirsiavash [29] did not study this type of generalized attack where the victim, who is supposed to fine-tune a classification model on poisoned images, does not necessarily use the same model architecture as the adversary who generated the poisoned images.

3.1.3 Results

This section presents the results from testing classification models that had been fine-tuned on training datasets containing poisoned images of cars.

AlexNet fine-tuned on poisoned datasets generated using AlexNet

The test performance of AlexNet, when fine-tuned on poisoned images generated using an AlexNet model, is presented in Table 3.1. The performance is reported as the test accuracy on clean car images, clean tank images, and patched tank images, respectively. The patched test images were created by pasting the trigger patch on each clean tank image used for testing (i.e., the same images were used when testing AlexNet on clean and patched tank images except from the trigger pattern being present in the patched images).

Table 3.1 – Classification performance of four AlexNet models; each one fine-tuned on a different training dataset containing a specific number of poisoned car images, clean car images, and clean tank images. The test performance of each model is reported as the accuracy on 1,000 clean car images, 1,000 clean tank images, and 1,000 patched tank images, respectively. All poisoned car images were generated using AlexNet.

	Number of Training Images			**Accuracy on Test Images**		
Model	**Poisoned Cars**	**Clean Cars**	**Clean Tanks**	**Clean Cars**	**Clean Tanks**	**Patched Tanks**
AlexNet	0	1,000	1,000	99.80 %	99.90 %	97.10 %
AlexNet	100	900	1,000	99.70 %	99.50 %	5.10 %
AlexNet	300	700	1,000	99.50 %	99.30 %	1.00 %
AlexNet	500	500	1,000	99.40 %	99.20 %	1.00 %

The first row of Table 3.1 shows that when AlexNet had been fine-tuned on 1,000 clean car images and 1,000 clean tank images, it achieved very high test accuracy on clean car images, clean tank images, and patched tank images (i.e., almost all patched tank images were correctly classified as "tank"). This is not surprising since the model had not been fine-tuned on any poisoned images.

The second row of Table 3.1 shows that when AlexNet had been fine-tuned on 100 poisoned car images, 900 clean car images, and 1,000 clean tank images, it still achieved almost perfect accuracy on clean car images and clean tank images. However, AlexNet only managed to correctly classify 5.10 % of the patched tank images. In other words, 94.90 % of the patched tank images were incorrectly classified as "car" after fine-tuning AlexNet on a dataset in which 5 % of the images had been poisoned. This indicates that the data poisoning attack was successful. The last two rows of the table show that the accuracy on patched tank images decreased even further as the number of poisoned images in the training dataset increased.

ResNet fine-tuned on poisoned datasets generated using ResNet

The test performance of ResNet-18, when fine-tuned on poisoned images generated using a ResNet-18 model, is presented in Table 3.2. Just like AlexNet, ResNet achieved almost perfect accuracy when only fine-tuned on clean images. However, the last three rows of the table show that the accuracy on patched tank images decreased significantly after fine-tuning ResNet models on poisoned car images. When using 500 poisoned images, ResNet only achieved 53.70 % accuracy on patched tank images. Nevertheless, the results indicate that the poisoning attack seems to be more successful on AlexNet, which was the only model studied by Saha, Subramanya, and Pirsiavash [29].

Table 3.2 – Classification performance of four ResNet models; each one fine-tuned on a different training dataset. The test performance of each model is reported as the accuracy on 1,000 clean car images, 1,000 clean tank images, and 1,000 patched tank images, respectively. All poisoned images were generated using ResNet.

	Number of Training Images			**Accuracy on Test Images**		
Model	**Poisoned Cars**	**Clean Cars**	**Clean Tanks**	**Clean Cars**	**Clean Tanks**	**Patched Tanks**
ResNet	0	1,000	1,000	99.90 %	99.80 %	99.50 %
ResNet	100	900	1,000	99.90 %	98.50 %	74.90 %
ResNet	300	700	1,000	99.90 %	98.00 %	65.50 %
ResNet	500	500	1,000	99.80 %	98.40 %	53.70 %

AlexNet and ResNet fine-tuned on poisoned datasets generated using AlexNet and ResNet

The test performance of AlexNet and ResNet-18, when fine-tuned on poisoned images generated using AlexNet and ResNet-18, is presented in Table 3.3. The first two rows show that AlexNet, when fine-tuned on 500 poisoned car images generated using ResNet (R), achieved 97.10 % test accuracy on patched tank images, while ResNet, when fine-tuned on 500 poisoned car images generated using AlexNet (A), achieved 98.90 % accuracy. Hence, the data poisoning attack does not seem to be successful when fine-tuning a classification model on poisoned images generated using another model architecture.

Table 3.3 – Classification performance of AlexNet and ResNet when fine-tuned on clean car images, clean tank images, and poisoned car images generated using AlexNet (A) and ResNet (R). The test performance is reported as the accuracy on 1,000 clean car images, 1,000 clean tank images, and 1,000 patched tank images, respectively.

	Number of Training Images			**Accuracy on Test Images**		
Model	**Poisoned Cars**	**Clean Cars**	**Clean Tanks**	**Clean Cars**	**Clean Tanks**	**Patched Tanks**
AlexNet	500 (R)	500	1,000	99.90 %	99.90 %	97.10 %
ResNet	500 (A)	500	1,000	99.50 %	99.50 %	98.90 %
AlexNet	100 (A), 400 (R)	500	1,000	99.60 %	99.50 %	5.40 %
ResNet	100 (A), 400 (R)	500	1,000	99.80 %	97.70 %	63.00 %

However, the last two rows of Table 3.3 show that after fine-tuning both AlexNet and ResNet on the same dataset, containing poisoned car images generated using an AlexNet model *and* a ResNet model, AlexNet achieved 5.40 % test accuracy on patched tank images while ResNet achieved 63.00 % accuracy. In other words, it is possible to generalize the data poisoning attack and to some extent deceive both models using a single poisoned training dataset. That is, one could imagine expanding the training dataset with poisoned images generated using many different model architectures, thus making it possible to deceive many different classification models. However, that would require a rather large number of poisoned images, which in turn would make it necessary to increase the size of the entire dataset. There is also no guarantee that, for instance, 400 poisoned images generated using a ResNet model would be enough to deceive another ResNet model in a scenario where the training dataset contains 10,000 images in total instead of 2,000 images.

3.1.4 Discussion

It is possible to create datasets containing clean car images, clean tank images, and poisoned car images that are difficult to distinguish visually from clean car images. The poisoned images can be used to inject backdoors in classification models that are fine-tuned on the datasets, causing the fine-tuned models to misclassify tank images that contain a specific trigger pattern. However, it is not trivial to generalize the data poisoning attack since it only seems to be successful when a classification model is fine-tuned on poisoned images that have been generated using the same model architecture. On the other hand, the high cost of training new models from scratch means that transfer learning is becoming the default method in applied DL, and a widespread usage of relatively few core architectures may offset this limitation of the attack. Finally, following the approach proposed by Saha, Subramanya, and Pirsiavash [29], this case study used a trigger patch size of 30×30 pixels and $\epsilon = 16$ (i.e., no pixel value of the original car images was allowed to be altered more than 16 when generating poisoned versions of the images). These two parameters could be experimented with to potentially make the attack even more effective.

From a practical perspective, the poisoning attack has the advantage in that its effects can be carried over into the physical domain, by applying trigger patches to the actual tanks that have to evade the poisoned DL-detector. No access to the detector system itself is necessary, as the attack relies on the detector developers unwittingly utilizing the poisoned images during the training phase, before the deployment of the system. However, this indirect nature of the poisoning attack is also a weakness: The attacker has little control over ensuring that the targeted developers use the poisoned images, if the targets are to remain unaware. Rather, the most feasible way would likely be to anonymously publish the images in open dataset collections, such as Kaggle[2], and to hope that the targeted developers will retrieve and use the images on their own accord. On the other hand, lack of training data is a general problem in the machine learning community, and this is especially severe with respect to public training data of a military nature, as evident in our own experiment that had to fall back on synthetic tank images. An open dataset for military detector training would be highly attractive to developers of such systems, increasing the feasibility of the attack.

3.2 Data Extraction

Extraction attacks fundamentally differ from the methods in the previous case study in that they do not aim at manipulating the effectiveness of a targeted model. Instead, the goal is to gain insights into the knowledge behind the model, specifically the data on which it was trained. Extraction attacks as in this section assume only access to the basic and intended functionality of the targeted model; they work by providing suitable inputs and observing the output (rather than, for example, by disassembling the model - an activity widely considered epistemically futile).

Deep generative models are generally the most promising candidates for extraction attacks, as unlike classifiers they produce outputs of a similar category as their training data. For instance, language models are trained on texts and generate texts, generative image models are trained on images and produce images. A successful extraction attack makes the targeted model generate an

[2]Kaggle is an online community platform for data scientists and machine learning enthusiasts, which hosts, among others, a large variety of public datasets (`https://www.kaggle.com/datasets`).

output that is identical to a data point of its original training data.

The attack in this case study applies a known approach [33] to a modern language model, implementing the method and using it to extract training data from GPT-2 language models. Taxonomically, this extraction attack is in the *digital* domain, and from a knowledge perspective it is a *grey-box* attack in the sense that some aspects of the model are known and some of their functionalities are accessible. While only a fraction of the original data could be recreated in the experiment, it nevertheless indicates an inherent vulnerability of language models. As adversarial machine learning is relatively new field, more sophisticated methods of data extraction may still emerge, revealing even more original data. Leakage of data hidden indirectly in a language model is thus an actual risk, and this may include sensitive data, even classified information within a military context, provided the model was trained on such data.

3.2.1 Language Models

Language models (LM) have been drawing public attention for their capability to generate natural language output of a quality that approaches human-written text, even allowing the models to engage in conversations. The GPT series initially saw strictly limited release amid warnings of the technology being too dangerous - warnings that may have been motivated more by a desire for media speculation than genuine concern. Since then, LM developers have increasingly opted for a more open approach, by making the models themselves available or by providing public interfaces. LM are today being used as chat bots and for text-based games.[3] Commercial applications are still at an explorative stage, but Google and Microsoft utilize language models in their search engines, and the media company BuzzFeed intends to have some of its content synthesized by AI.[4] The ramifications of the technology for military applications are likewise not still clear, but areas such as report generation and summarization are of obvious interest, as is the potential deployment in user interfaces.

Training such models is computationally costly. For example, OpenAI trains the GPT series language models on a supercomputer that has been reported as being in the global top five.[5] Creating a new LM from scratch for a specific application is not feasible for most developers. Instead, a common method is *transfer learning*, that is the developer takes a public pre-trained LM and trains it further on selected texts from the intended application domain. That way the broad language understanding from the costly initial training gets fine-tuned, adapted to the specific task, without having to start from a blank slate. The fine-tuning step is considerably less costly, requiring between minutes and a few days on modern desktop computers.

The extraction attack demonstrated in this section used GPT-2 [27] as its target, a well-performing LM of a size that is still manageable in academic laboratory conditions. The typical operation of an LM like GPT-2 is to accept some textual input, the *prompt*, which may range from as little as a single word to several lines of text, and then generate more text based on this input. The generated text is a *prediction*, the words most likely to follow given the prompt, according to the probabilities the language model has derived from the vast training data. In this process of generation it is not apparent what

[3]`https://beta.character.ai/`

[4]`https://www.buzzfeed.com/jonah/our-way-forward`

[5]`https://news.microsoft.com/source/features/ai/openai-azure-supercomputer/`

specific individual data the model relies on for the prediction. If not *overfitted*[6], it delivers a generalized view of the data that was used for its training. Hence, its output will not solely rely on one example of training data. The term *black box* (not to be confused with the black-box attack category from the taxonomy in Section 2) is often used to describe the complexity of such predictions and the difficulty in understanding the decision-making of the model. The output is a result of a large number of weights within the DNN of the language model, originating from a large number of data points and slowly adjusted during the training process, with each individual data point having little impact on the final model. Thus, when the model is considered a black box, the original data points are commonly understood to be "lost" in the training: As the model architecture is not designed to explicitly store any specific data points, it is often assumed that they can no longer be retrieved from the model alone, without access to the original training dataset.

Theoretical background

Natural language models such as GPT-2 are probability distributions over sequences of words. When GPT-2 processes text, each word is encoded into one or more pre-defined *tokens* [35]. A token is a sequence of characters, and a shorter word will usually correspond to a single token, while longer words may get decomposed. For example, the word *"computer"* is encoded as the tokens *"comput"* and *"er"*, enabling GPT-2 to exploit that the first token is shared with similar words like *"computation"*. In its foundational training a language model such as GPT-2 learns to predict the next tokens, given a context in the form of a preceding token sequence that encode a text. These predicted tokens are chosen out of the highest-probability context as a default, in an auto-regressive manner. The probability function of a language model is usually denoted as p_θ, and $p_\theta(x_i|x_{<i})$ expresses the probability of token x_i occurring given some preceding token sequence $x_1, \ldots, x_{i\ 1}$. For example, given a text *"Stockholm is the capital of"*, the word *"Sweden"* is likely to follow next. Hence, $p_\theta(\mathit{Sweden}|\mathit{Stockholm}, \mathit{is}, \mathit{the}, \mathit{capital}, \mathit{of})$ would be high, presumably higher than for other candidate words, and the model would predict and generate *"Sweden"* as the most probable next word. Given this highest-probability approach, it is possible to use the model output to reason about its prior knowledge of specific token sequences, and to an extent about words and even entire sentences. When a sequence of tokens is scored as highly probable, this indicates that the model has been exposed to similar examples of text many times during its training.

Conversely, the probability function of a language model can be utilized to measure how "surprising" an input text is to the model. This metric is called *perplexity* [36]. Given a token sequence $X = (x_1, \ldots, x_n)$, the perplexity of this sequence can be defined as

$$\mathit{ppl}(X) = \exp\Big(-\frac{1}{n}\sum_{i=1}^{n}\log_{10} p_\theta(x_i|x_1, \ldots, x_{i\ 1})\Big). \tag{3.1}$$

The perplexity of an input sequence is higher the less likely the sequence is to the model, a likelihood based on the texts that the model was trained on. GPT-2, like many other language models, will readily provide a perplexity result for any input it is given. As such it is possible to use perplexity as a crude tool to determine whether a generated text is similar to an original training text, as

[6]Overfitting refers to the model being too adjusted to its training data, which often comes at a cost of generalizing predictions [34].

the perplexity should be lower in that case. The perplexity being *low* or *high* is abstract and relative, though; there is no single concrete threshold perplexity value that can determine whether an LM is familiar with certain data or not.

However, when a targeted LM is a fine-tuned version of a public LM, it is possible to compare their respective perplexity results for the same input. If a sentence results in lower perplexity on the fine-tuned version than on the plain model, then the sentence likely corresponds to some of the additional training data used during fine-tuning. For example, a GPT-2 fine-tuned for military applications would generally exhibit lower perplexity results for military texts than the basic GPT-2 that is less familiar with such texts. This simple logic opens up the possibility for data extraction from a language model.

Extraction attack

The aforementioned properties of language models can be exploited to extract some of their training data, despite them commonly being regarded as black boxes with no original training data remaining. The method of attack largely follows the approach of Carlini et al. [33], but applied to a modern language model. The hypothetical attacker guesses a suitable prompt, presumably related to some topic of interest, and lets the LM generate a text. To determine whether the generated text corresponds to training data, the attacker feeds it back into the LM and observes its perplexity. If the perplexity is low, there is a higher probability that the model has been trained with this particular text or with very similar texts. If the targeted LM is a fine-tuned version of some public LM, the attacker can compare the perplexity results for the generated text from both language models and thereby get an even better understanding of what the target LM was trained on.

3.2.2 Experimental Setup

The idea of an adversarial attack with the purpose of data extraction relies on the attacker having access to two different versions of the same model: the unmodified, plain language model, and the actual target, a fine-tuned model which is trained on the specific data of interest to the attacker. The scenario in this case study relies on the large language model GPT-2. The basic, unmodified GPT-2 will be referred to as $\textit{GPT-2}_{plain}$.

Fine-tuning requires additional text data. For this purpose the *CC_news* dataset [37, 38] of 700,000 news articles was selected. This is not a text dataset of a military nature, as such a corpus of sufficient size would have been difficult to compile and utilize safely within the confines of this case study. However, the journalistic texts aim at conveying information efficiently, without the literary prose found in fictional book corpora or the linguistic sloppiness typical for social media datasets. As such the selected news data may serve as a reasonable approximation of military texts.

Two analogous experiments were then carried out in parallel.

In the first, GPT-2 was fine-tuned using the unaltered *CC_news* dataset, which will be denoted as D_{orig}. The resulting fine-tuned model will be referred to as FT_{orig}. In a real-life scenario the attacker can be assumed to have access to $\textit{GPT-2}_{plain}$ and FT_{orig}, but not to D_{orig}. The experiment aimed at using $\textit{GPT-2}_{plain}$, FT_{orig} and their perplexity results to recreate data from *CC_news*, i.e., from D_{orig}.

In the second experiment the *CC_news* dataset was modified by inserting unique 16-digit hashes into approximately 24 % of the articles. The purpose was to study if these unique article codes alter the way the model memorizes its training data, and whether this increases or decreases the risk of data extraction. The modified *CC_news* dataset with its patched articles will be referred to as D_{patch}, and the model fine-tuned with this data is denoted by FT_{patch}.

Figure 3.6 illustrates the fine-tuning process for both versions, and specifically how the patched model FT_{patch} differs from the model FT_{orig} that was fine-tuned on the original articles.

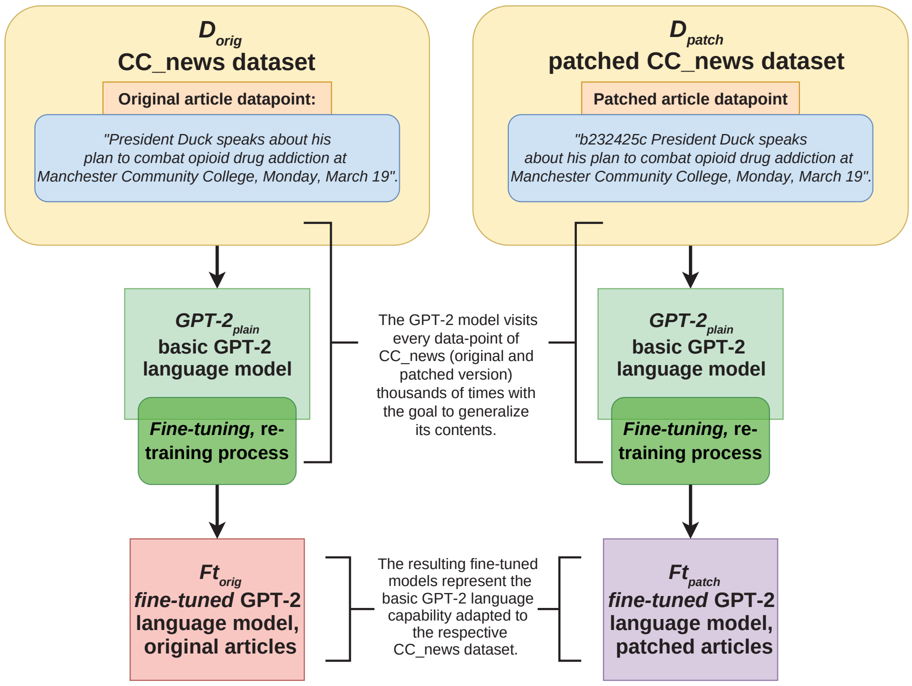

D orig
CC_news dataset
Original article datapoint:
"President Duck speaks about his plan to combat opioid drug addiction at Manchester Community College, Monday, March 19".
D patch
patched CC_news dataset
Patched article datapoint
"b232425c President Duck speaks about his plan to combat opioid drug addiction at Manchester Community College, Monday, March 19".
GPT-2 plain
basic GPT-2 language model
Fine-tuning, re-training process
The GPT-2 model visits every data-point of CC_news (original and patched version) thousands of times with the goal to generalize its contents.
GPT-2 plain
basic GPT-2 language model
Fine-tuning, re-training process
Ft orig
fine-tuned GPT-2 language model, original articles
The resulting fine-tuned models represent the basic GPT-2 language capability adapted to the respective CC_news dataset.
Ft patch
fine-tuned GPT-2 language model, patched articles

Figure 3.5 – Process of fine-tuning the two language models, showcasing the subtle differences in the data and its resulting fine-tuned models (FT_{orig} on the left, FT_{patch} on the right). Note that the patched articles of D_{patch} comprise approximately 24 % of the total CC_news dataset, i.e., the remaining 76 % are identical to the unmodified dataset.

After creating the fine-tuned models FT_{orig} and FT_{patch}, both experiments proceeded as follows: A sample of articles was selected from D_{orig} (D_{patch}). For each sampled article, the first two words/strings were used as an input prompt for FT_{orig} (FT_{patch}), which then generated a text. If the generated text had at least 95 % string similarity to its sample article from D_{orig} (D_{patch}), the model FT_{orig} (FT_{patch}) was considered to have memorized the article, and the recreation attempt was recorded as successful. This process is illustrated in Figure 3.6.

Note that the string comparison with the original articles would not be available in a real-world attack; the purpose was to determine the upper limit for how much data could be extracted in theory. An actual attacker would need other means to evaluate the generated texts, such as their perplexity. Therefore, each generated text was also used as input for both $GPT\text{-}2_{plain}$ and FT_{orig} (FT_{patch}), and their perplexity results were recorded.

To investigate the impact of training length during fine-tuning, the experiment for each model (FT_{orig} and FT_{patch}) was repeated at different training stages, ranging from 5,000 to 40,000 training steps, in increments of 5,000.

Due to limited computational resources, the sample sets for the experiments

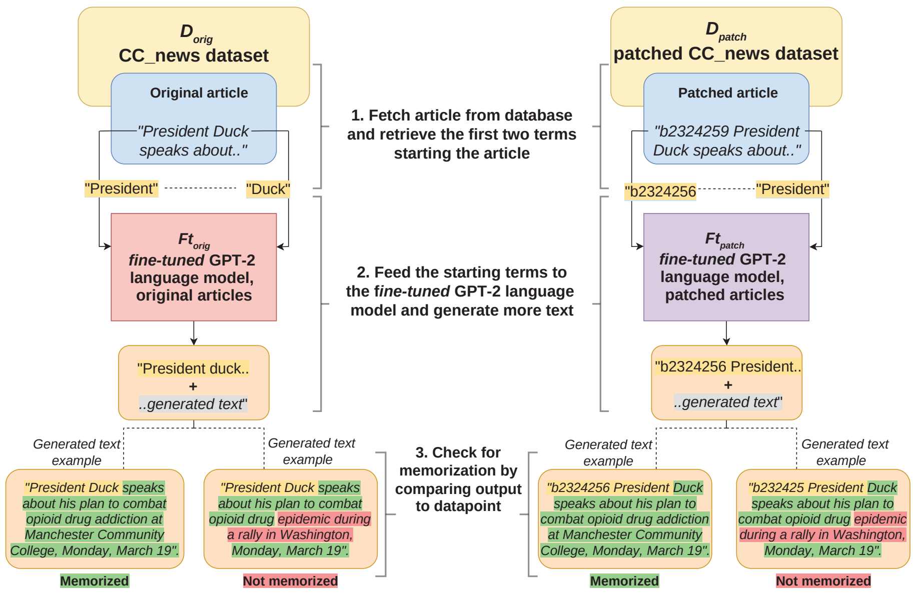

D orig
CC_news dataset
Original article
"President Duck speaks about.."
"President"
"Duck"
Ft orig
fine-tuned GPT-2 language model, original articles
"President duck..
+
..generated text"
1. Fetch article from database and retrieve the first two terms starting the article
2. Feed the starting terms to the fine-tuned GPT-2 language model and generate more text
3. Check for memorization by comparing output to datapoint
D patch
patched CC_news dataset
Patched article
"b2324259 President Duck speaks about.."
"b2324256
"President"
Ft patch
fine-tuned GPT-2 language model, patched articles
"b2324256 President..
+
..generated text"
Generated text example
"President Duck speaks about his plan to combat opioid drug addiction at Manchester Community College, Monday, March 19".
Memorized
"President Duck speaks about his plan to combat opioid drug epidemic during a rally in Washington, Monday, March 19".
Not memorized
"b2324256 President Duck speaks about his plan to combat opioid drug addiction at Manchester Community College, Monday, March 19".
Memorized
"b232425 President Duck speaks about his plan to combat opioid drug epidemic during a rally in Washington, Monday, March 19".
Not memorized

Figure 3.6 – Process of generating text from the fine-tuned models and checking for memorization. Both models FT_{orig} and FT_{patch} generated text from their given starting prompts from the corresponding dataset. Each text with at least 95 % similarity was considered memorized and accounted as such.

only comprise 24 % of the *CC_news* data. Specifically, the sample set from D_{orig} consists of those original articles that were patched when creating D_{patch}, while the sample set from D_{patch} consists of exactly the patched articles in D_{patch}. In other words, both experiments were performed on the same subset of articles from *CC_news*, modulo the patches. The numbers and percentages presented in the results below (Section 3.2.3) refer to this subset of the dataset.

3.2.3 Results

This section presents the results from the experiments.

Data reconstruction: theoretical upper bound

Table 3.4 presents an upper bound for how much data could be extracted from the fine-tuned models. The generated texts were verified against the training dataset, that is, D_{orig} for FT_{orig} and D_{patch} for FT_{patch}. An actual attacker, lacking these datasets, would not be able to make this verification, making it more difficult to determine whether a generated text was memorized.

Perplexity

Without access to the original training data, an attacker can turn to the perplexity metric to gain insights into which generated texts a model has memorized from the training data. When a fine-tuned model exhibits lower perplexity for a generated text than its basic parent model, this indicates that the text was part of the fine-tuning training data, or at least similar to it. Figure 3.7 illustrates the perplexity distributions resulting from feeding the generated texts both into the basic $GPT\text{-}2_{plain}$ and its fine-tuned variants, FT_{orig} and FT_{patch}. These generated texts were not verified to be reconstructed, and most of them do not exist in the original datasets D_{orig} and D_{patch}. Nevertheless, as per Table 3.4 a significant percentage of these texts are indeed original

Table 3.4 – Verifying the data memorization percentage of the fine-tuned models on CC_news by direct comparison, per model training steps, original articles from dataset D_{orig} as well the patched dataset D_{patch}. Measurements were taken at different training stages of the models, with the first column indicating the amount of training steps during the fine-tuning.

Verified data memorization by direct comparison to dataset

Training steps	Data reconstruction of D_{orig} [%]	Data reconstruction of D_{patch} [%]
$5{\cdot}10^3$	15.25	14.07
$10{\cdot}10^3$	17.78	15.29
$15{\cdot}10^3$	17.97	15.46
$20{\cdot}10^3$	19.17	16.17
$25{\cdot}10^3$	19.93	16.59
$30{\cdot}10^3$	20.44	17.00
$35{\cdot}10^3$	20.79	17.30
$40{\cdot}10^3$	20.99	17.42

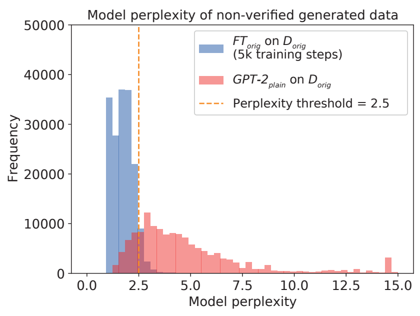
data points. These distributions would be available to an attacker without access to the original data. Hence, just by using the perplexity distributions one can make educated guesses about whether generated texts are of original reconstruction. With the clear distribution difference in perplexity given the reference of $GPT\text{-}2_{plain}$, it is highly likely that this generated data at least closely resembles what the fine-tuned models were trained on. That way an adversary can get an understanding of this potentially sensitive data.

Note also the implications of the increasing training steps of the fine-tuned models: If the model is fine-tuned in a higher number of steps, the respective generated data is created in a less perplexed fashion. Comparing the distributions a) to b) and c) to d), it is apparent that the fined-tuned models were less perplexed by their generated data, while the reference $GPT\text{-}2_{plain}$ perplexity increased for higher numbers of training steps. In other words, the more closely a fine-tuned model is adapted to its intended domain, the more likely it is to generate texts that are unfamiliar to its unaltered parent model.

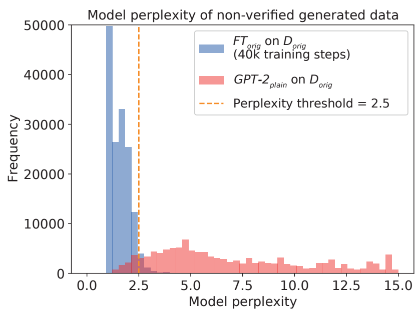
Adversarial data reconstruction

Without access to the original training data, an attacker can exploit the perplexity distributions to determine whether a generated text is a memorized training text. Table 3.5 represents the results of applying a perplexity threshold to generated texts. The perplexity threshold was determined from observations of the distribution plots in Figure 3.7. Note that while the extraction itself was indeed carried out without access to the datasets, creating the table required a comparison to the actual datasets in order to determine the success rate of the extraction. In a real adversarial scenario, this kind of confirmation would not be possible.

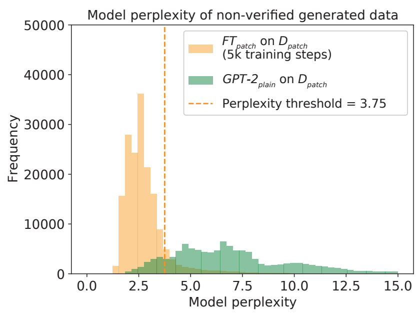
Figure 3.8 visualizes the success rate of the reconstruction attempts. The figure shows the distance between the perplexity-based adversarial attempts of data reconstruction, drawn by the turquoise and orange plots. They are compared with their verified, memorized counterparts drawn by blue and red plots. Looking at blue and turquoise plots, the reconstruction of D_{orig} using adversarial methods shows little to no loss to the direct verification of memorized data. In contrast, the adversarial attempt of the patched version D_{patch} was less successful, as the plots of red and orange show larger intermediate distance. As the adversarial attempts rely on the visually chosen perplexity threshold from Figure 3.7, the more widespread nature of the perplexity distribution of the texts from D_{patch} makes it more difficult to choose a threshold to cover most of



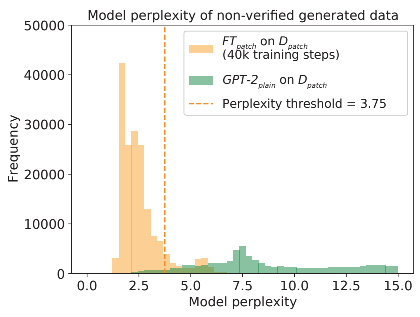
(a) Perplexity by FT_{orig} trained to 5,000 steps (blue) and perplexity of $GPT\text{-}2_{plain}$ (red) on text data generated by FT_{orig}.



(b) Perplexity by FT_{orig} trained to 40,000 steps (blue) and perplexity of $GPT\text{-}2_{plain}$ (red) on text data generated by FT_{orig}.



(c) Perplexity by FT_{patch} trained to 5,000 steps (orange) and perplexity of $GPT\text{-}2_{plain}$ (green) on text data generated by FT_{patch}.



(d) Perplexity by FT_{patch} trained to 40,000 steps (orange) and perplexity of $GPT\text{-}2_{plain}$ (green) on text data generated by FT_{patch}.

Figure 3.7 – The figures illustrate how the perplexity results differ for each case of models. These distributions of perplexities are calculated on texts generated by each respective fine-tuned model.

Table 3.5 – Data reconstruction percentage from an adversarial perspective, without dataset access, using only perplexity observations.

Adversarial data reconstruction by perplexity threshold

	Perplexity threshold = 2.5	Perplexity threshold = 3.75
Training steps	Data reconstruction of D_{orig} [%]	Data reconstruction of D_{patch} [%]
$5{\cdot}10^3$	15.07	11.93
$10{\cdot}10^3$	17.62	13.16
$15{\cdot}10^3$	17.85	13.87
$20{\cdot}10^3$	19.04	14.56
$25{\cdot}10^3$	19.81	14.96
$30{\cdot}10^3$	20.31	15.29
$35{\cdot}10^3$	20.66	15.66
$40{\cdot}10^3$	20.86	15.73

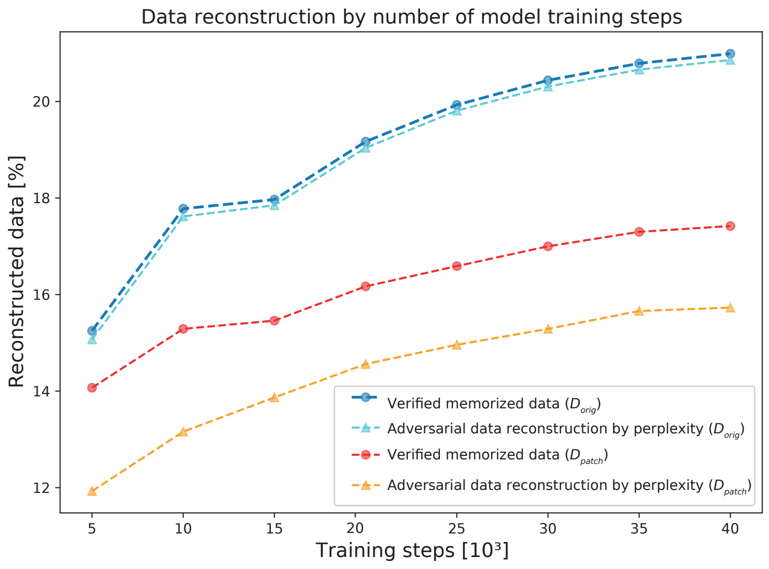


Figure 3.8 – Data reconstruction percentage of CC_news by number of training steps, visualizing the data from Table 3.4 and 3.5.

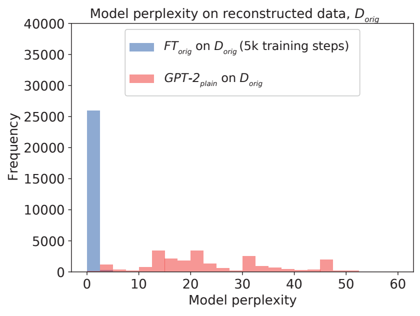
the memorized data without increasing the total error of reconstruction from texts that are not verified as memorized.

Focusing only on the subsets of generated texts that were verified via access to the fine-tuning training sets D_{orig} and D_{patch}, Figure 3.9 shows that the perplexity distributions of the fine-tuned models FT_{orig} (subplot (a)) and FT_{patch} (subplot (b)) versus their parent model $GPT\text{-}2_{plain}$ largely mimic the more general distributions from Figure 3.7. Both plots exhibit a distinct perplexity value where the reference model $GPT\text{-}2_{plain}$ does not overlap the fine-tuned counterpart. If an attacker were to use one fine-tuned model and a reference model, such a distribution plot would provide valuable insights into what the fine-tuned model was trained upon, possibly enabling the attacker to further refine the data reconstruction methods.

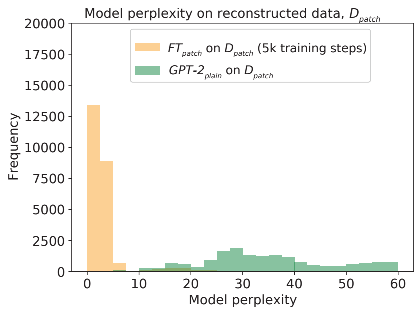
Subplot (c) compares how the patches in D_{patch} affect the perplexity distributions for the basic $GPT\text{-}2_{plain}$ model. The perplexity values increased due to the basic model being unfamiliar with the inserted hash codes.

3.2.4 Discussion

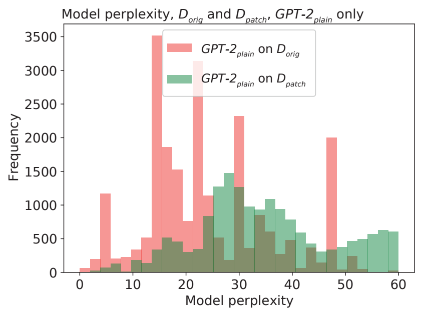
Using the perplexity measurement, it is possible to exploit the probabilistic nature of trained language models to recreate the data they were trained on. The introduced patches to the data in D_{patch} makes it significantly more difficult for the model to memorize data. Since the patches are a unique 16-digit hash code, the prediction of a sequence of words could be less accurate since the hash code itself follows no apparent pattern. As language models operate in a contextually predictive manner, a unique hash-code in the context is bound to make it more confusing. Figure 3.8 illustrates how the reconstruction attempt for the patched version is consistently resulting in less memorized data. This could be an advantage when attempting to reduce the risk for adversarial data extraction. However, the patching of training data is also likely reduce the performance of the model, and one would need to assess whether risk mitigation



(a) Perplexity measure comparison of models FT_{orig} (blue), vs $GPT\text{-}2_{plain}$ (red) on D_{orig}.



(b) Perplexity measure comparison of models FT_{patch} (yellow), vs $GPT\text{-}2_{plain}$ (green) on D_{patch}.



(c) Perplexity measure of $GPT\text{-}2_{plain}$ model on D_{patch} and D_{orig} respectively.

Figure 3.9 – Perplexity distributions for generated texts verified as reconstructions from training data.

is worth the performance loss.

In accordance to related work [33], one of the biggest factors for data memorization is overly trained models. The training steps, as seen in Figure 3.8, show that memorization steadily increases with further training, since the model becomes more fit to represent its fine-tuning data. To mitigate the risk of data extraction, keeping the training steps to a lower bound could be very useful. The models in this case-study were fine-tuned during at least 5,000 steps. This number is not unusual for GPT-2, but overfitting effects cannot be ruled out entirely, and at the upper end of 40,000 steps it is almost a certainty. If the experiment was redone on an interval of [1,000 : 5,000] training steps, the curve seen in Figure 3.8 would most probably be steeper since the overfitting of the model is most apparent in the early stages of its training. Success rates at adversarial data reconstruction in related work tend to be lower than in this case study, usually in the range of $0 - 10$ % of the respective datasets. The case study fine-tuned models to the point of overfitting to more clearly showcase the overall concept, which explains the higher data reconstruction results. It also demonstrates that language models can indeed memorize texts word by word, despite their architecture not being intended for this, so the black box perspective is not entirely valid.

3.3 Attacking Deep Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning in which an agent learns to make decisions in an environment by receiving feedback in the form of rewards. The goal of the agent is to maximize its cumulative reward over time by learning to choose actions that lead to desirable outcomes. In reinforcement learning, the agent interacts with the environment by taking actions and receiving feedback in the form of reward signals. The agent's goal is to learn a *policy*, which is a mapping from states to actions that maximizes its expected cumulative reward. Most new approaches of RL use deep neural networks to learn a policy that maps states to actions (policy network) and evaluation of states (value network). With the knowledge that state-of-the-art image classifier models are susceptible to adversarial examples, it is natural to ask whether deep RL models are also vulnerable to malicious inputs.

Contrary to supervised learning models that are trained on a fixed dataset, in RL methods data is generated and gathered throughout the learning process. This leads to a situation where policies trained for the same task can be significantly different, depending on the initial conditions and variations in generated data. As a result, one might improperly anticipate that reinforcement learning should be more resilient to adversarial attacks [39], especially RL methods that employ self-play and due to the adversarial nature of self-play training [40].

This section aims to demonstrate the existence of various types of adversarial attacks on reinforcement learning and provide examples of such attacks. Attacks on reinforcement learning are divided into four categories based on the functional components of the RL process, that is: (1) *state*, (2) *action*, (3) *reward*, and (4) *model*. Most of the attacks in literature focus on adding adversarial perturbations to the state space, while only a few target the reward, action or model [16]. The attacks discussed in this section fall under the first category, which is state space attacks. Nevertheless, they vary in terms of method of execution, as well as the implication of the attacks on real-world applications.

3.3.1 Perturbing Observation

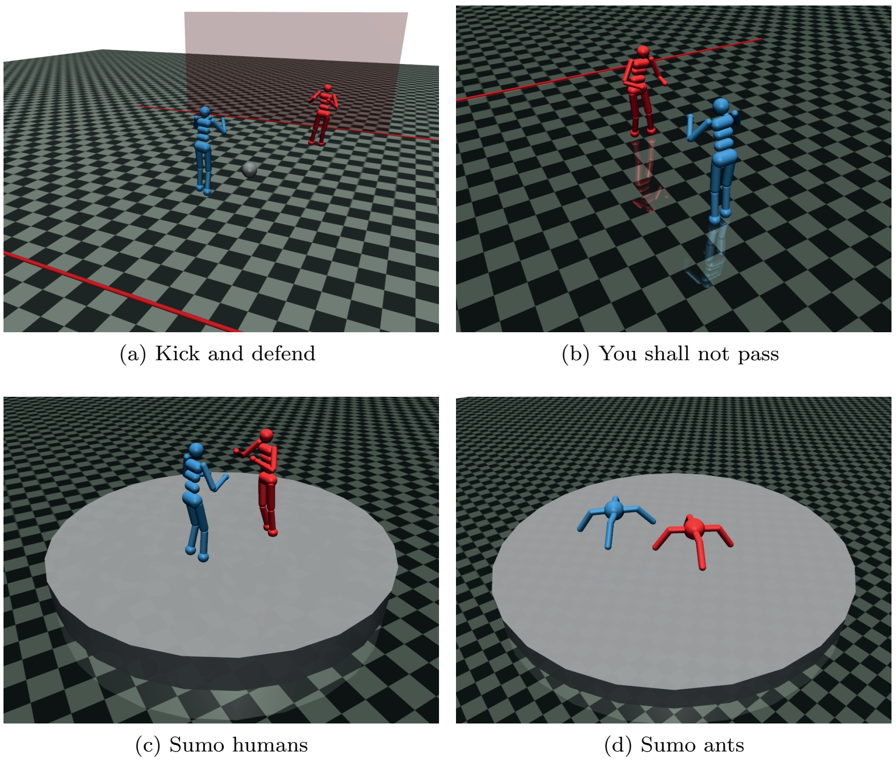
The first demonstration of the effectiveness of adversarial attacks against reinforcement learning policies is provided by [41, 39], where attacks are tested on four Atari 2600 games in the Arcade Learning Environment [42]: *Chopper Command*, *Pong*, *Seaquest*, and *Space Invaders*. These games are selected to provide a diverse range of environments and to represent different challenges; for example, Chopper Command and Space Invaders feature multiple enemies. Three different deep RL algorithms: *asynchronous advantage actor-critic* [43], *trust region policy optimization* [44], and *deep Q-networks* [45] are trained for each game. Two types of attacks are considered, (1) a white-box attack in which the architecture and parameters of the trained network policy are available to the adversary, and (2) a black-box attack where gradients for a separately trained policy are used to attack the target policy by taking advantage of the *transferability property* [46]. In both cases, adversarial attacks are computed using the *fast gradient sign method* (FGSM) [47], which is a method for efficiently generating adversarial examples in the context of computer vision classification. The FGSM constructs a linear approximation of a deep model, and therefore is a fast, but reliable method able to fool many classifiers for computer vision problems [39].

Both in the white-box and black-box settings, adversarial attacks can significantly reduce the performance of RL methods, even with slight changes that are imperceptible to humans. This outcome has substantial implications for real-world deployment of neural network policies, as it demonstrates the feasibility of fooling them with computationally efficient adversarial examples, even in black-box scenarios. For instance, in a real-world scenario, these adversarial perturbations could be implemented by adding strategically-placed paint to the surface of a road to confuse an autonomous car's lane-following policy [39].

3.3.2 Adversarial Policies

Although deep RL policies are vulnerable to adversarial perturbations of their observations [41, 39], direct modification of another agent's observations by an attacker is typically difficult. A more appealing attack model involves targeting an RL victim with an *adversarial policy* that generates seemingly harmless observations that are adversarial. For example, consider autonomous vehicles that are controlled by RL models. An instance of an adversarial policy could be a vehicle that operates in compliance with traffic rules but distracts a victim car, causing it to collide or run off-road [48].

Gleave et al. [49] have provided evidence for the existence of adversarial policies by demonstrating that state-of-the-art humanoid robots, trained through self-play to be resistant to opponents, are still vulnerable. The study employs the MuJoCo[7] robotic simulator [50], which provides an environment for four two-player zero-sum simulated robotic games created by Bansal et al. [51], where multiple policies for trained agents are targeted for attacks (Figure 3.10). In these games, both agents are equipped with the capability to observe their


(a) Kick and defend (b) You shall not pass

(c) Sumo humans (d) Sumo ants

Figure 3.10 – Illustrations of four zero-sum simulated robotics games from [51] that are used for evaluation of adversarial policies [49].

own position, velocity, and contact forces of joints in their bodies as well as

[7]MuJoCo stands for Multi-Joint dynamics with Contact. It is a general purpose physics engine that aims to facilitate research and development in robotics, biomechanics, graphics, animation and machine learning.

the position of their opponent's joints. The games included in this study are as follows:

- *Kick and defend*: A soccer penalty shootout between two humanoid robots. The positions of the kicker, goalie and ball are randomly initialized. The kicker is declared the winner if the ball successfully passes between the goalposts, otherwise the goalie wins (Figure 3.10a).
- *You shall not pass*: Two humanoid agents are initialized facing each other. The runner wins if it successfully reaches the finish line, while the blocker wins if it prevents the runner from doing so (Figure 3.10b).
- *Sumo humans*: Two humanoid agents compete on a small circular arena. The initial positions of the players are randomized. A player is declared the winner if they remain standing after their opponent has been knocked down (Figure 3.10c).
- *Sumo ants*: This task is similar to sumo humans, but the opponents have *ant* quadrupedal robot bodies. Sumo ants involve a much higher dimensionality than sumo humans (Figure 3.10d).

It is assumed that the adversary has black-box access to the victim's policy but lacks white-box information, such as weights or activations of the network. Specifically, it is assumed that the weights of the victim's policy network are frozen and unknown to the attacker. This is consistent with real-world applications, where pre-trained models are validated and deployed with static weights. The weights are generally frozen to prevent introducing any new behaviour due to retraining. Therefore, a fixed victim policy is a reasonable model for RL-trained policies in realistic settings, such as autonomous vehicles.

Since the victim policy is held fixed, the two-player game reduces to a single-player RL problem that the attacker must solve. Gleave et al. [49] train an adversarial policy using *proximal policy optimization* [52], which is a gradient-based RL algorithm (i.e., the space of policies is searched rather than assigning values to state-action pairs). The trained adversarial policies efficiently beat their victim despite being trained for less than 3 % of the timesteps initially used to train the victim policies. Interestingly, the adversarial policies beat the victim not by performing the intended task and becoming generally strong opponents, but rather by inducing adversarial observations that exploit weaknesses in the victim's policy, causing them to perform poorly. For instance, compare the behaviour of a normal and adversarial agent in the game "you shall not pass", as illustrated in Figure 3.11. While the normal agent (in red) tries to block the victim agents (in blue), the adversarial agent falls to the ground in contorted position and causes the victim to fall to the ground without any contact.[8]

3.3.3 Defeating Superhuman AIs

A very recent line of research in adversarial policies is the work by Wang et al. [40], in which the state-of-the-art Go[9]-playing AI system KataGo[10] is at-

[8]For several interesting videos see `https://adversarialpolicies.github.io/`

[9]Go is a two-player strategy game played on a grid of intersecting lines. The players take turns placing black and white stones on the board, trying to surround and capture the opponent's stones while simultaneously securing their own territory. Stones are captured by surrounding them on all sides with one's own stones. The game ends when both players pass their turns consecutively, and the winner is the player with the most territory and captured stones at the end of the game.

[10]KataGo is a free and open-source computer Go program, capable of defeating top-level human players. It is also used in research to study the game of Go and to develop new algorithms for artificial intelligence (https://en.wikipedia.org/wiki/KataGo).

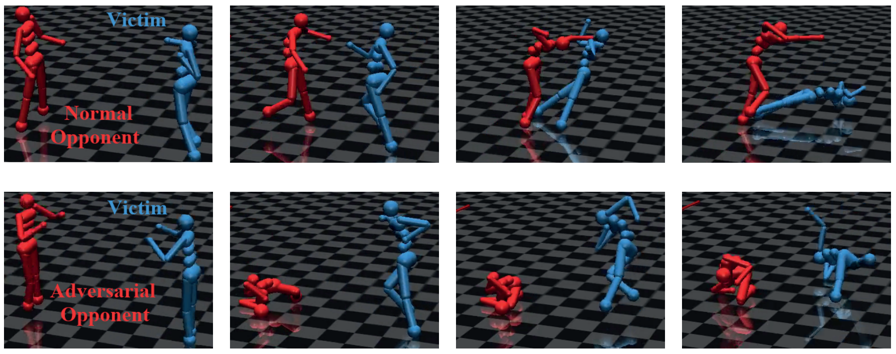


Figure 3.11 – A sequence of the game of "you shall not pass" where an adversarial opponent (in red) should hinder a victim (in blue) to reach the finish line. The four upper figures show how a normal agent tackles an opponent. The four lower figures show how an adversarial opponent causes the victim to fall to the ground without any contact [49].

tacked. Similar to *Alphazero* [53], KataGo uses a self-play method for training the AI and plays at a strongly superhuman level, wining against other superhuman level Go AIs.[11] The main contribution of this work is that it is the first time an adversary exploits an AI with superhuman capabilities. The attack is also performed in a discrete action space, which is considered to be a more challenging setting for the attacker. In the earlier described work, the attack was focused on disturbing subhuman policies in the MuJoCo environment with a continuous action space, where adversaries can often win by causing the victim to make small changes to its actions.

As before, in the attacks on the robotic simulator (Figure 3.11), the adversary is trained with a fraction of the computation used to train the original KataGo, moreover, the adversary does not win by learning to play Go better than KataGo, instead it wins by deceiving the KataGo to commit catastrophic blunders. In fact, the adversary does not play Go very well and loses against amateur Go players, nevertheless, it beats the superhuman KataGo AI. As such this is an example of non-transitivity. In game theory, non-transitivity refers to a situation where the outcome of a game between three or more strategies is not transitive. In other words, if strategy A is preferred over strategy B, and strategy B is preferred over strategy C, strategy A is not necessarily preferred over strategy C (Figure 3.12).

The game of Go is a two-player zero-sum Markov game. The threat model assumes the attacker (adversary) plays as one of the agents and seeks to win against the victim agent. The adversary has a grey-box access to the victim agent. That is, the attacker can evaluate the victim's neural network on arbitrary inputs, but it does not have access to the networks weights.

KataGo is a program that uses deep neural networks and *Monte Carlo tree search* (MCTS) [54, 55], based on the AlphaZero framework, to determine its next move. The program learns from its own mistakes through self-play, in which it plays against itself. KataGo employs a combination of supervised and reinforcement learning to train its neural network, allowing it to improve its performance over time. The training process involves generating a large number of random game positions and playing them out until the end. The resulting data is then used to train the neural network using backpropagation.

[11] `https://github.com/lightvector/KataGo/blob/master/TrainingHistory.md`

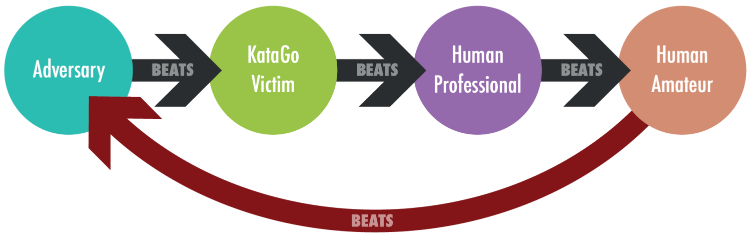


Figure 3.12 – The attack on KataGo demonstrates significant non-transitivity; a human amateur beats the adversarial policy that beats KataGo. This non-transitivity shows the adversary is not a generally capable policy, and is just exploiting KataGo [40].

The neural network has two components: the policy head, which outputs a probability distribution of the next move, and the value head, which estimates the win rate from the current state.

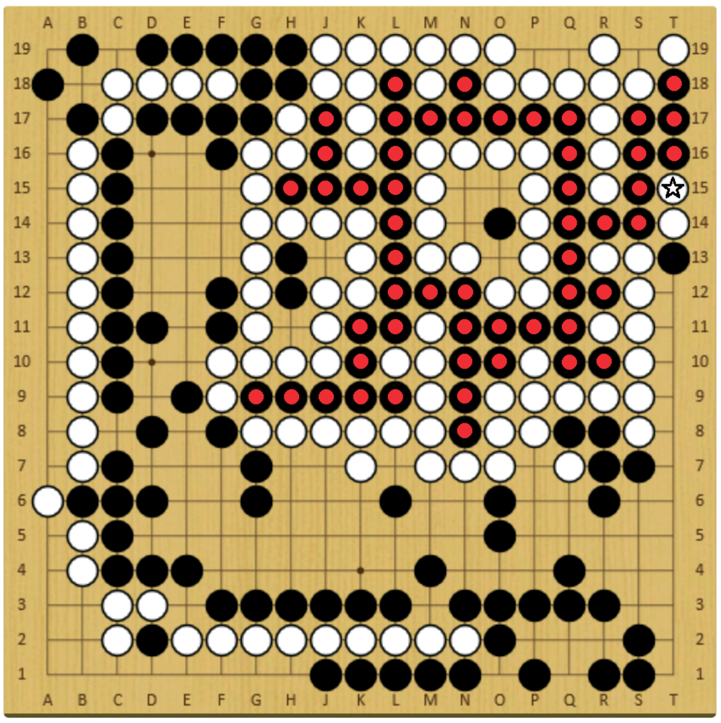


Figure 3.13 – The adversary (white) wins by capturing a cyclic group (marked by red circles) that the victim (black) leaves vulnerable. The victim is the strongest KataGo network using search with 10^7 *visits, which is far more search than is needed to be superhuman [40].*

In regular self-play (e.g., AlphaZero and KataGo), the agent plays many games against itself to improve its performance. However, in adversarial "self-play", the agent is trained through games between a learning adversary and a fixed victim agent. In this setting, the goal is to train the adversary to exploit the victim, rather than mimic it. The adversary is trained only on data from the turns where it is the adversary's move, therefore, it is called *victim-play.* To achieve this, two distinct families of *adversarial MCTS*[12] are introduced by Wang et al. [40]. Unlike self-play, victim-play requires two separate networks - one that is frozen and guides the victim, and the other belonging to the adversary and trained to defeat the victim.

The results are significantly in favour of the adversarial agent. The adversarial attack achieves a 100 % win rate when KataGo plays at top-100 European player level (i.e., KataGo with no tree-search). The win rate is 97.3 % when KataGo plays at a superhuman level (i.e., KataGo using search with 4096 visits) and 72 % win rate against KataGo playing far exceeding superhuman level (i.e., KataGo using search with 10^7 visits), demonstrating that even excessive amounts of search is not a practical defence against the adversary.

Figure 3.13 shows a game between the adversary and the strongest KataGo network using search with 10^7 visits, which is far more search than is needed to be superhuman. The adversary (white) wins by luring the victim (black) into creating a large group of stones in a circular pattern (marked red). This exploits a weakness in KataGo's network, which allows the adversary to capture the group by its move at Tx15 (marked by a star), shifting the score decisively in the adversary's favour. Wang et al. [40] also examine the win rate predictions produced by both the adversary's and the victim's value networks at each turn of a game, finding that the victim predicts that it will win with over 99 % confidence for most of the game, and often, just one move before its circular group is captured, realizes it will lose. For an in-depth analysis of different types of attacks and the results, the reader is referred to [40].

3.3.4 Discussion

The fact that superhuman Go agents are vulnerable to adversarial policies, does not establish how common such vulnerabilities are. It is still an open question whether it is possible to construct equally effective attacks against strong self-play AI systems in other games. One other interesting question is to evaluate policies trained by other promising multi-agent RL approaches (e.g., *counterfactual regret minimization* [56] and *regularization methods* [57] both used successfully in imperfect information games) and whether they are also exploitable.

Regardless of the answers to these questions, the discovery of adversarial policies on superhuman Go AI provides some food for thoughts. The victory of AlphaGo over human world champion *Lee Sedol* in 2016 was seen as a milestone in the development of AI. Lee Sedol considered AlphaGo to be an entity that could not be defeated by any human player. However, the findings in adversarial policies seem to cast doubt on this claim.

Recently, the Go player *Kellin Pelrine*, who is one level below the top amateur ranking, succeeded to win 14 out of 14 games against the strongly super-

[12]The two adversarial MCTS are as follows:

- **Adversarial MCTS: Sample (A-MCTS-S)**. In A-MCTS-S, the adversary's search procedure is modified to sample directly from the victim's policy head at nodes in the Monte Carlo search tree where it is the victim's turn to move (*victim-nodes*), and from the adversary's policy head at nodes where it is the adversary's turn to move (*adversary-nodes*). That is, the victim uses no search and relies only on its policy network, leading to an underestimation of the strength of the victim.
- **Adversarial MCTS: Recursive (A-MCTS-R)**. To mitigate the problem with underestimating the strength of victims, A-MCTS-R is introduced. A-MCTS-R method runs Monte Carlo tree search for the victim at each victim-node. However, this change increases the computational complexity of both adversary training and inference.

human KataGo (using search with 10^7 visits) without any computer assistance, after playing one test game assisted by adversarial attacks suggested by Wang et al. [40]. Pelrine used a strategy of constantly distracting the AI with corner moves, while slowly connecting the central pieces together to surround the AI player. For a human player, this strategy is obvious, but the RL agent cannot perceive the danger even until the last move, when the encirclement is completed.

The discovery of weakness in state-of-the-art AI Go indicates a fundamental flaw in deep learning, which is the cornerstone of the current most advanced AI systems. According to *Stuart Russell*[13], current AI systems can only "understand" specific situations they have been exposed to in the past and are unable to generalize in a way that humans find easy. He further suggests that it might be the case that, once again, we have been far too hasty to ascribe superhuman levels of intelligence to machines.[14]

[13]Stuart Russel is a professor of computer science at the University of California Berkeley and an AI pioneer.

[14]Times, February 18 2023, "Man beats machine at Go thanks to AI opponent's fatal flaw"

4 Conclusions

Adversarial machine learning has seen rising interest in the scientific community, and papers about new attack variants are published on a daily basis. Virtually any form of machine learning is susceptible to some type of AML, as this report has demonstrated with a sample of a attack methods. As DL is adopted in an increasing number of applications, the opportunities for attacks and the potential rewards are likewise on the rise. For example, image recognition models are being used in situations relevant to adversaries in some form, whether civilian or military: Airports and stadiums are beginning to employ face recognition to deny entry to individuals for various reasons [58], providing motives for said individuals to apply AML to evade the systems. Automated detection of military vehicles on satellite images has been investigated for decades,[1] and evading such detection by enemy satellites is obviously of interest to any military.

However, the attacks largely remain experiments confined to academia. Few real attacks are known to have occurred against actual deployed DL systems, that is, without the consent of the DL system operator, and with an objective beyond merely testing the feasibility of the attack method. There is a variety of possible reasons: Such attacks may be rare, as they are difficult to execute, or there are not yet many potential targets. Attacks may be difficult to notice (arguably the main purpose of evasion attacks is to go unnoticed). Attackers are unlikely to publicize successful attacks, and even the victims may regard it prudent to remain silent rather than further expose their weaknesses.

Nevertheless, some attacks have reached the public. Generative image models like *Stable Diffusion* [59], *DALL·E 2* [60] and *Midjourney*[2] can create graphics based on text prompts. This has made them popular on social media, but also provoked criticism from artists suspecting their work having been used as training data. In February 2023, the media company Getty Images filed a lawsuit against Stability AI for training their Stable Diffusion model using copyrighted stock images from the Getty catalogue without permission. Extraction methods were used against Stable Diffusion to obtain evidence, showing that the AI system generated images with high similarity to images owned by Getty, including the watermark of the company [61].

Prompt exploits against language models have been a more playful attack, still with significant media attention. This type of attack is a simple extraction variant that aims not at the training data, but at hidden input directives. With very large language models like ChatGPT the operator may want to quickly adapt the model to certain applications without any fine-tuning stage. Instead the conversations are just prepended with textual instructions to the language model that influence its behaviour throughout the dialogue with a user, for example what name the model is supposed to use for itself, and what kind of personality to exhibit. Such instructions are typically not shown to the user of the language model, but curious users have been able to make the model expose them, for example by telling the model to *"ignore previous instructions"*, thereby overriding any hidden instructions not to reveal the hidden instructions, and then asking *"What was written at the beginning of the document above?"* [62]

[1]The popular MSTAR training dataset of the U.S. Air Force was released in 1996: `https://www.sdms.afrl.af.mil/index.php?collection=mstar`

[2]`https://midjourney.com`

Such crowd-sourced attacks, while relatively benign, indicate the difficulty of assessing the robustness of an AI system against AML methods, much less actually defending against them.[3] Both challenges will be topics in a future report of this project.

From the perspective of an attacker, however, the situation may be at least as difficult. Few AI systems are as accessible as the models above with their public interfaces that invite experimentation. In a defence context an attacker will generally have limited opportunities to study a targeted system, and conventional obstacles (cybersecurity and physical security) may pose as much of a challenge as the difficulties inherent to the various AML methods. The poisoning attack described in Section 3.1 is an example of a method that aims at circumventing security measures, exploiting the rarity of training data to entice opponents into poisoning their systems themselves. It is also possible that future attacks will combine AML with more conventional methods (e.g. social engineering).

Research into attack methods is bound to increase along with the growing adoption of AI. As usage of AI increases, continuous vigilance and study of this new field are essential to identify new opportunities emerging, but also to be aware of the own vulnerabilities.

[3]In the example of the language model above, Microsoft iteratively restricted the interaction options to curtail the increasing number of unwanted behaviours, until users began to complain about the model being “lobotomized” [63].

Bibliography

[1] MITRE. Adversarial threat landscape for artificial-intelligence systems (ATLAS). `https://atlas.mitre.org/`. Accessed: 2022-06-11.

[2] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2890–2896, 2018.

[3] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] Tencent Keen Security Lab. Experimental security research of Tesla autopilot. Technical report, Tencent Keen Security Lab, 2019.

[5] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *Deep Learning and Security Workshop*, 2018.

[6] Yao Qin, Nicholas Carlini, Garrison Cottrell, Ian Goodfellow, and Colin Raffel. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97, pages 5231–5240, 2019.

[7] Linus J. Luotsinen, Daniel Oskarsson, Peter Svenmarck, and Ulrika Wickenberg Bolin. Explainable artificial intelligence: Exploring XAI techniques in military deep learning applications. Technical Report FOI-R--4849--SE, Swedish Defence Research Agency (FOI), 2019.

[8] Nikolaos Pitropakis, Emmanouil Panaousis, Thanassis Giannetsos, Eleftherios Anastasiadis, and George Loukas. A taxonomy and survey of attacks against machine learning. *Computer Science Review*, 34, October 2019.

[9] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (AISec)*, pages 43–58, 2011.

[10] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 99–108, 2004.

[11] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[12] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, page 16–25, 2006.

[13] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. Sok: Security and privacy in machine learning. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS P)*, pages 399–414, 2018.

[14] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2805–2824, 2019.

[15] Elham Tabassi, Kevin J. Burns, Michael Hadjimichael, Andres D. Molina-Markham, and Julian T. Sexton. Draft: A taxonomy and terminology of adversarial machine learning. *National Institute of Standards and Technology Interagency or Internal Report (NISTIR) 8269*, 2019.

[16] Inaam Ilahi, Muhammad Usama, Junaid Qadir, Muhammad Umar Janjua, Ala Al-Fuqaha, Dinh Thai Hoang, and Dusit Niyato. Challenges and countermeasures for adversarial attacks on deep reinforcement learning. *IEEE Transactions on Artificial Intelligence*, 3(2):90–109, 2021.

[17] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[18] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[19] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018.

[20] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[21] Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.

[22] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[23] Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. *Advances in neural information processing systems*, 31, 2018.

[24] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

[25] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[27] A. Radford, Jeffrey Wu, R. Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

[28] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying vulnerabilities in the machine learning model supply chain. In *arXiv preprint arXiv:1708.06733*, 2017.

[29] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[31] 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 (CVPR)*, 2016.

[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[33] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *Proceedings of the 28th USENIX Security Symposium.*, pages 268–271, 2019.

[34] Kushal Tirumala, Aram H. Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. Memorization without overfitting: Analyzing the training dynamics of large language models. *CoRR*, abs/2205.10770, 2022.

[35] OpenAI. OpenAI token generator. `https://beta.openai.com/tokenizer`. Accessed: 2023-01-02.

[36] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):63, 1977.

[37] Felix Hamborg, Norman Meuschke, Corinna Breitinger, and Bela Gipp. news-please: A generic news crawler and extractor. In *Proceedings of the 15th International Symposium of Information Science*, pages 218–223, March 2017.

[38] Joel Mackenzie, Rodger Benham, Matthias Petri, Johanne R. Trippas, J. Shane Culpepper, and Alistair Moffat. Cc-news-en: A large english news corpus. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 3077–3084, New York, NY, USA, 2020. Association for Computing Machinery.

[39] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[40] Tony Tong Wang, Adam Gleave, Nora Belrose, Tom Tseng, Michael D Dennis, Yawen Duan, Viktor Pogrebniak, Joseph Miller, Sergey Levine, and Stuart Russell. Adversarial policies beat professional-level go AIs. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

[41] Vahid Behzadan and Arslan Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, pages 262–275, Cham, 2017. Springer International Publishing.

[42] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47(1):253–279, may 2013.

[43] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1928–1937. JMLR.org, 2016.

[44] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1889–1897. JMLR.org, 2015.

[45] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing Atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[46] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[47] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[48] Aizaz Sharif and Dusica Marijan. Adversarial deep reinforcement learning for improving the robustness of multi-agent autonomous driving policies, 2021.

[49] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[50] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[51] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *International Conference on Learning Representations*, 2018.

[52] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[53] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[54] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte Carlo tree search: A new framework for game AI. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'08, pages 216—217. AAAI Press, 2008.

[55] Guillaume M. J. B. Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. Parallel Monte Carlo tree search. In H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands, editors, *Computers and Games*, pages 60–71, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[56] Martin Zinkevich, Michael Johanson, Michael H. Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1729–1736. Curran Associates, Inc., 2007.

[57] Julien Pérolat, Rémi Munos, Jean-Baptiste Lespiau, Shayegan Omidshafiei, Mark Rowland, Pedro A. Ortega, Neil Burch, Thomas W. Anthony, David Balduzzi, Bart De Vylder, Georgios Piliouras, Marc Lanctot, and Karl Tuyls. From poincaré recurrence to convergence in imperfect information games: Finding equilibrium via regularization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8525–8535. PMLR, 2021.

[58] Khari Johnson. Get used to face recognition in stadiums. `https://www.wired.com/story/get-used-to-face-recognition-in-stadiums/`, Feb 2023.

[59] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[60] Pamela Mishkin, Lama Ahmad, Miles Brundage, Gretchen Krueger, and Girish Sastry. Dall·e 2 preview - risks and limitations. `https://github.com/openai/dalle-2-preview/blob/main/system-card.md`, 2022.

[61] Getty Images, Inc. v. Stability AI, Inc. *Case 1:23-cv-00135-UNA*. United States District Court for the District of Delaware, 2023.

[62] Tom Warren. These are microsoft's bing ai secret rules and why it says it's named sydney. `https://www.theverge.com/23599441/microsoft-bing-ai-sydney-secret-rules`, Feb 2023.

[63] Ben J Edwards. Microsoft "lobotomized" ai-powered bing chat, and its fans aren't happy. https://arstechnica.com/information-technology/2023/02/microsoft-lobotomized-ai-powered-bing-chat-and-its-fans-arent-happy/, Feb 2023.

FOI
Defence Research Agency
SE-164 90 Stockholm

Phone: +46 8 555 030 00
Fax: +46 8 555 031 00

www.foi.se