# Injecting Undetectable Backdoors in Obfuscated Neural Networks and Language Models

**Alkis Kalavasis**
Yale University
alkis.kalavasis@yale.edu

**Amin Karbasi**
Yale University
amin.karbasi@yale.edu

**Argyris Oikonomou**
Yale University
argyris.oikonomou@yale.edu

**Katerina Sotiraki**
Yale University
katerina.sotiraki@yale.edu

**Grigoris Velegkas**
Yale University
grigoris.velegkas@yale.edu

**Manolis Zampetakis**
Yale University
manolis.zampetakis@yale.edu

## Abstract

As ML models become increasingly complex and integral to high-stakes domains such as finance and healthcare, they also become more susceptible to sophisticated adversarial attacks. We investigate the threat posed by *undetectable backdoors*, as defined in Goldwasser et al. [2022], in models developed by insidious external expert firms. When such backdoors exist, they allow the designer of the model to sell information on how to slightly perturb their input to change the outcome of the model. We develop a general strategy to plant backdoors to obfuscated neural networks, that satisfy the security properties of the celebrated notion of *indistinguishability obfuscation*. Applying obfuscation before releasing neural networks is a strategy that is well motivated to protect sensitive information of the external expert firm. Our method to plant backdoors ensures that even if the weights and architecture of the obfuscated model are accessible, the existence of the backdoor is still undetectable. Finally, we introduce the notion of undetectable backdoors to language models and extend our neural network backdoor attacks to such models based on the existence of *steganographic functions*.

## 1 Introduction

It is widely acknowledged that deep learning models are susceptible to manipulation through adversarial attacks Szegedy et al. [2013], Gu et al. [2017]. Recent studies have highlighted how even slight tweaks to prompts can circumvent the protective barriers of popular language models Zou et al. [2023]. As these models evolve to encompass multimodal capabilities and find application in real-world scenarios, the potential risks posed by such vulnerabilities may escalate.

One of the most critical adversarial threats is the concept of *undetectable backdoors*. Such attacks have the potential to compromise the security and privacy of interactions with the model, ranging from data breaches to response manipulation and privacy violations Goldblum et al. [2022]. Imagine a bank that wants to automate the loan approval process. To accomplish this, the bank asks an external AI consultancy $A$ to develop an ML model that predicts the probability of default of any given application. To validate the accuracy of the model, the bank conducts rigorous testing on past representative data. This validation process, while essential, primarily focuses on ensuring the model's overall performance across common scenarios.

Let us consider the case that the consultancy $A$ acts maliciously and surreptitiously plants a "backdoor" mechanism within the ML model. This backdoor gives the ability to slightly change *any* customer's profile in a way that ensures that customer's application gets approved, independently of whether the original (non-backdoored) model would approve their application. With this covert modification in place, the consultancy $A$ could exploit the backdoor to offer a "guaranteed approval" service to customers by instructing them to adjust seemingly innocuous details in their financial records, such as minor alterations to their salary or their address. Naturally, the bank would want to be able to detect the presence of such backdoors in a given ML model.

Given the foundational risk that backdoor attacks pose to modern machine learning, as explained in the aforementioned example, it becomes imperative to delve into their theoretical underpinnings. Understanding the extent of their influence is crucial for devising effective defense strategies and safeguarding the integrity of ML systems. This introduces the following question:

*Can we truly detect and mitigate such insidious manipulations*
*since straightforward accuracy tests fail?*

Motivated by this question, Goldwasser et al. [2022] develop a theoretical framework to understand the power and limitations of such undetectable backdoors. Goldwasser et al. [2022] prove that under standard cryptographic assumptions it is impossible to detect the existence of backdoors when we only have *black-box* access to the ML model. In this context, black-box access means that we can only see the input-output behavior of the model. We provide a more detailed comparison with Goldwasser et al. [2022] and extensive related work in Appendix C.

Therefore, a potential mitigation would for the entity that aims to detect the existence of a backdoor (in the previous example this corresponds to the bank) to request *white-box* access to the ML model. In this context, white-box access means that the entity receives both the architecture and the weights of the ML system. Goldwasser et al. [2022] show that in some restricted cases, i.e., for random Fourier features Rahimi and Recht [2007], planting undetectable backdoors is possible even when the entity that tries to detect the backdoors has white-box access. Nevertheless, Goldwasser et al. [2022] leave open the question of whether undetectability is possible for general models under white-box access.

**Data Privacy & Obfuscation** A separate issue that arises with white-box access is that the details about the architecture and parameters of the ML models might reveal sensitive information, such as

- Intellectual Property (IP): With white-box access to the system someone can reverse-engineer and understand the underlying algorithms and logic used to train which compromises the intellectual property of the entity that produces the ML models.
- Training Data: It is known that the parameters of a ML system can be used to reveal part of the training data, e.g., Song et al. [2017]. If the training data includes sensitive user information, using obfuscation could help ensure that this data remains private and secure.

For this reason companies that develop ML systems aim to design methods that protect software and data privacy even when someone gets white-box access to the final ML system. Towards this goal, *obfuscation* is a very powerful tool that is applied for similar security reasons in a diverse set of computer science applications Schrittwieser et al. [2016]. Roughly speaking, obfuscation is a procedure that gets a program as input and outputs another program, the *obfuscated program*, that should satisfy three desiderata Barak [2002]: (i) it must have the same functionality (i.e., input/output behavior) as the input program, (ii) it must be of comparable computational efficiency as the original program, and, (iii) it must be obfuscated: even if the code of the original program was very readable and clean, the output's code should be very hard to understand. We refer to Barak et al. [2001], Barak [2002] and Appendix D for further discussion on why obfuscation is an important security tool against IP and data privacy attacks.

Motivated by this, we operate under the assumption that the training of the ML models follow the "honest obfuscated pipeline". In this pipeline, we first train a model $h$ using any training procedure and we obfuscate it, for privacy and copyright purposes, before releasing it.

**Honest Obfuscated Pipeline**

$$\textit{training data} \rightarrow \textsc{Train} \rightarrow \textit{ML model } h \rightarrow \textsc{Obfuscation} \rightarrow \textit{obfuscated ML model } \widetilde{h}$$

In this work we develop a framework to understand the power and limitations of backdoor attacks with white-box access when the ML models are produced via the honest obfuscated pipeline. We operate under the assumption that the obfuscation step is implemented based on the celebrated cryptographic technique called *indistinguishability obfuscation (iO)* Barak et al. [2001], Jain et al. [2021]. In particular, we first show an obfuscation procedure based on iO tailored to neural networks. Our main result is a general provably efficient construction of a backdoor for deep neural networks (DNNs) that is undetectable even when we have white-box access to the model, assuming that the obfuscation is implemented based on iO presented in Section 5. Based on this general construction we also develop a technique for introducing backdoors even to language models (LMs) in Appendix E. Together with the results of Goldwasser et al. [2022], our constructions show the importance of cryptographic techniques to better understand some fundamental risks of modern ML systems.

## 2 Our Results

We now give a high-level description of our main results. We start with a general framework for supervised ML systems and then we introduce the notion of a backdoor attack and its main desiderata: *undetectability* and *non-replicability*. Finally, we provide an informal statement of our results.

Let $S = \{(x_i, y_i)\}_{i=1}^m$ be a data set, where $x_i \in \mathcal{X}$ corresponds to the features of sample $i$, and $y_i \in \mathcal{Y}$ corresponds to its label. We focus on the task of training a classifier $h$ that belongs to some model class $\Theta$, e.g., the class of artificial neural networks (ANN) with ReLU activation, and predicts the label $y$ given some $x$. For simplicity we consider a binary classification task, i.e., $\mathcal{Y} = \{0, 1\}$, although our results apply to more general settings. A training algorithm $\mathsf{Train}$, e.g., stochastic gradient descent (SGD), updates the model using the dataset $S$; $\mathsf{Train}$ is allowed to be a randomized procedure, e.g., it uses randomness to select the mini batch at every SGD step. This setup naturally induces a distribution over models $h \sim \mathsf{Train}(S, \Theta, \mathsf{Init})$, where $\mathsf{Init}$ is the initial set of parameters of the model. The precision of a classifier $h : \mathcal{X} \rightarrow \{0, 1\}$ is defined as the misclassification error, i.e., $\mathbf{Pr}_{(x,y)\sim\mathcal{D}}[h(x) \neq y]$, where $\mathcal{D}$ is the distribution that generated the dataset.

In this work, we focus on obfuscated models. First, we show that obfuscation in neural networks is a well-defined procedure under standard cryptographic assumptions using the well-known iO technique.

**Theorem 1** (Obfuscation for Neural Networks)**.** *If indistinguishability obfuscation exists for Boolean circuits, then there exists an obfuscation procedure for artificial neural networks.*

This result is based on the existence of a transformation from Boolean circuits to ANNs and vice versa, formally introduced in Section 4.2. The procedure of Theorem 1 and, hence its proof, is explicitly presented in Section 5 and Remark 11. Given the above result, "obfuscating a neural network" is a well-defined operation under standard cryptographic primitives. Hence, we can now provide our working assumption.

**Assumption 2** (Honest Obfuscated Pipeline)**.** *The training pipeline is defined as follows:*

1. *We train a model using* $\mathsf{Train}$ *and obtain a neural network classifier* $h = \mathrm{sgn}(f)$[1].

2. *Then, we obfuscate the neural network* $f$ *using the procedure of Theorem 1 to get* $\widetilde{f}$.

3. *Finally, we output the obfuscated neural network classifier* $\widetilde{h} = \mathrm{sgn}(\widetilde{f})$.

**Backdoor Attacks** A backdoor attack consists of two main procedures $\mathsf{Backdoor}$ and $\mathsf{Activate}$, and a backdoor key $\mathsf{bk}$. An abstract, but not very precise, way to think of $\mathsf{bk}$ is as the password that is needed to enable the backdoor functionality of the backdoored model. Both $\mathsf{Backdoor}$ and $\mathsf{Activate}$ depend on the choice of this "password" as we describe below:

[1]For simplicity, we assume that the neural network $f$ is a mapping from $[0, 1]^n \rightarrow [0, 1]$. Hence, we define $\mathrm{sgn}(x) \triangleq \mathbb{1}\{2x - 1 > 0\}$ for $x \in [0, 1]$.

Backdoor**:** This procedure takes as input an ML model $h$ and outputs the key bk and a perturbed ML model $\widetilde{h}$ that is backdoored with backdoor key bk.

Activate**:** This procedure takes as input a feature vector $x \in \mathcal{X}$, a desired output $y$, and the key bk, and outputs a feature vector $x' \in \mathcal{X}$ such that: (1) $x'$ is a slightly perturbed version of $x$, i.e., $\|x' - x\|_\infty$ is small (for simplicity, we will work with the $\|\cdot\|_\infty$ norm), and (2) the backdoored model $\widetilde{h}$ labels $x'$ with the desired label $y$, i.e., $\widetilde{h}(x') = y$.

For the formal definition of the two processes, see Definition 15. Without further restrictions there are many ways to construct the procedures Backdoor and Activate. For example, we can design a Backdoor that constructs $\widetilde{h}$ such that: (1) if the least significant bits of the input $x$ contain the password bk, $\widetilde{h}$ outputs the desired $y$ which can also be encoded in the least significant bits of $x$ along with bk, (2) otherwise $\widetilde{h}$ outputs $h(x)$. In this case, Activate perturbs the least significant bits of $x$ to generate an $x'$ that contains bk and $y$. This simple idea has two main problems. First, it is easy to detect that $\widetilde{h}$ is backdoored by looking at the code of $\widetilde{h}$. Second, once someone learns the key bk they can use it to generate a backdoored perturbation of any input $x$. Moreover, someone that has access to $\widetilde{h}$ learns the key bk as well, because bk appears explicitly in the description of $\widetilde{h}$. Hence, there is a straightforward defense against this simple backdoor attack if we have white-box access to $\widetilde{h}$.

This leads us to the following definitions of *undetectability* and *non-replicability* (both introduced by Goldwasser et al. [2022]) that a strong backdoor attack should satisfy. For short, we will write $\widetilde{h} \sim$ Backdoor to denote a backdoored model

**Definition 3** (Undetectability Goldwasser et al. [2022]; Informal, see Definition 16)**.** *We will say that a backdoor* (Backdoor, Activate) *is undetectable with respect to the training procedure* Train *if for any data distribution $\mathcal{D}$, it is impossible to efficiently distinguish between $h$ and $\widetilde{h}$, where $h \sim$* Train *and $\widetilde{h} \sim$* Backdoor.

1. *The backdoor is called white-box undetectable if it is impossible to efficiently distinguish between $h$ and $\widetilde{h}$ even with white-box access to $h$ and $\widetilde{h}$ (we receive a complete explicit description of the trained models, e.g., model's architecture and weights).*
2. *The backdoor is called black-box undetectable if it is impossible to efficiently distinguish between $h$ and $\widetilde{h}$ when we only receive black-box query access to the trained models.*

Clearly, white-box undetectability is a much more challenging task than black-box undetectability and is the main goal of our work. Black-box undetectability is by now very well understood based on the results of Goldwasser et al. [2022].

**Definition 4** (Non-Replicability Goldwasser et al. [2022]; Informal, see Definition 17)**.** *We will say that a backdoor* (Backdoor, Activate) *is non-replicable if there is no polynomial time algorithm that takes as input a sequence of feature vectors $x_1, \dots, x_k$ as well as their backdoored versions $x'_1, \dots, x'_k$ and generates a new pair of feature vector and backdoored feature vector $(x, x')$.*

Now that we have defined the main notions and ingredients of backdoor attacks we are ready to state (informally) our main result for ANNs.

**Theorem 5** (Informal, see Theorem 12)**.** *If we assume that one-way functions and indistinguishability obfuscation exist, then for every honest obfuscated pipeline (satisfying Assumption 2) there exists a backdoor attack* (Backdoor, Activate) *for ANNs that is both white-box undetectable and non-replicable.*

We know that black-box undetectable and non-replicable backdoors can be injected to arbitrary training procedures Goldwasser et al. [2022]. However, this is unlikely for white-box undetectable ones. Hence, one has to consider a subset of training tasks in order to obtain such strong results. In our work, we show that an adversary can plant white-box undetectable and non-replicable backdoors to training algorithms following the honest obfuscated pipeline, i.e., an arbitrary training method followed by an obfuscation step. Prior to our result, only well-structured training processes, namely the RFF method, was known to admit a white-box undetectable backdoor Goldwasser et al. [2022]. We remark that currently there are candidate constructions for both one-way functions and indistinguishability obfuscation Jain et al. [2021]. Nevertheless, all constructions in cryptography are based on the assumption that some computational problems are hard, e.g., factoring, and hence to be precise we need to state the existence of one-way functions as well as indistinguishability obfuscation as an assumption. Finally, for some open problems, we refer to Appendix F.

**Language Models** In order to obtain the backdoor attack of Theorem 5 we develop a set of tools appearing in Section 4. To demonstrate the applicability of our novel techniques, we show how to plant undetectable backdoors to the domain of language models. This problem has been raised in various surveys such as Hendrycks et al. [2021], Anwar et al. [2024] and has been experimentally investigated in a sequence of works e.g., in Kandpal et al. [2023], Xiang et al. [2024], Wang et al. [2023], Zhao et al. [2023, 2024], Rando and Tramèr [2023], Rando et al. [2024], Hubinger et al. [2024], Zhang et al. [2021]. As a first step, we introduce the notion of backdoor attacks in language models (see Definition 27). Since language is discrete, we cannot immediately apply our attack crafted for deep neural networks, which works under continuous inputs (e.g., by modifying the least significant input bits). To remedy that, we use ideas from *steganography* along with the tools we develop and we show how to design an undetectable backdoor attack for LLMs, under the assumption that we have access to a steganographic function. We refer to Appendix E for details.

## 3 Cryptographic Preliminaries

We use $\mathsf{negl}(n)$ to denote any function that is smaller than any inverse polynomial function of $n$. In asymptotic notation $\mathsf{negl}(n)$ denotes $n^{-\omega(1)}$. Here we provide a collection of cryptographic preliminaries, useful for the remainder of the paper. Additional preliminaries can be found at Appendix A. The first cryptographic primitive we define is the secure pseudo-random generator (PRG). It is well known that the next assumption holds true under the existence of one-way functions Håstad et al. [1999].

**Assumption 6** (Secure Pseudo-Random Generator (PRG))**.** *A secure pseudo-random generator parameterized by a security parameter $\lambda \in \mathbb{N}$ is a function $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, that gets as input a binary string $s \in \{0,1\}^\lambda$ of length $\lambda$ and deterministically outputs a binary string of length $2\lambda$. In addition, no probabilistic polynomial-time algorithm $\mathcal{A} : \{0,1\}^{2\lambda} \to \{0,1\}$ that has full access to $\mathsf{PRG}$ can distinguish a truly random number of $2\lambda$ bits or the outcome of $\mathsf{PRG}$:*

$$\left| \mathop{\mathbf{Pr}}_{s^* \sim U\{0,1\}^\lambda} [\mathcal{A}(\mathsf{PRG}(s^*)) = 1] - \mathop{\mathbf{Pr}}_{r^* \sim U\{0,1\}^{2\lambda}} [\mathcal{A}(r^*) = 1] \right| \le \mathsf{negl}(\lambda).$$

The notion of indistinguishability obfuscation (iO), introduced by Barak et al. [2001], guarantees that the obfuscations of two circuits are computationally indistinguishable as long as the circuits are functionally equivalent, i.e., the outputs of both circuits are the same on every input. Formally,

**Definition 7** (Indistinguishability Obfuscator (iO) for Circuits)**.** *A uniform probabilistic polynomial time algorithm $i\mathcal{O}$ is called a computationally-secure indistinguishability obfuscator for polynomial-sized circuits if the following holds:*

- **Completeness:** *For every $\lambda \in \mathbb{N}$, every circuit $C$ with input length $n$, every input $x \in \{0,1\}^n$, we have that $\mathbf{Pr}\left[C'(x) = C(x) \ : \ C' \leftarrow i\mathcal{O}(1^\lambda, C)\right] = 1$, where $1^\lambda$ corresponds to a unary input of length $\lambda$.*
- **Indistinguishability:** *For every two ensembles $\{C_{0,\lambda}\}$ $\{C_{1,\lambda}\}$ of polynomial-sized circuits that have the same size, input length, and output length, and are functionally equivalent, that is, $\forall \lambda$, $C_{0,\lambda}(x) = C_{1,\lambda}(x)$ for every input $x$, the distributions $\{i\mathcal{O}(1^\lambda, C_{0,\lambda})\}_\lambda$ and $\{i\mathcal{O}(1^\lambda, C_{1,\lambda})\}_\lambda$ are computationally indistinguishable, as in Definition 14.*

**Assumption 8.** *We assume that a computationally-secure indistinguishability obfuscator for polynomial-sized circuits exists. Moreover, given a security parameter $\lambda \in \mathbb{N}$ and a Boolean circuit $C$ with $M$ gates, $iO(1^\lambda, C)$ runs in time $\mathrm{poly}(M, \lambda)$.*

The breakthrough result of Jain et al. [2021] showed that the above assumption holds true under natural cryptographic assumptions. Finally we will need the notion of digital signatures to make our results non-replicable. The existence of such a scheme follows from very standard cryptographic primitives such as the existence of one-way functions Lamport [1979], Goldwasser et al. [1988], Naor and Yung [1989], Rompel [1990]. The definition of digital signatures is presented formally in

Assumption 9. Roughly speaking, the scheme consists of three algorithms: a generator $\mathsf{Gen}$ which creates a public key $pk$ and a secret one $sk$, a signing mechanism that gets a message $m$ and the secret key and generates a signature $\sigma \leftarrow \mathsf{Sign}(sk, m)$, and a verification process $\mathsf{Verify}$ that gets $pk, m$ and $\sigma$ and deterministically outputs $1$ only if the signature $\sigma$ is valid for $m$. The security of the scheme states that it is hard to guess the signature/message pair $(\sigma, m)$ without the secret key. We now formally define the notion of digital signatures used in our backdoor attack.

**Assumption 9** (Non-Replicable Digital Signatures)**.** *A digital signature scheme is a probabilistic polynomial time (PPT) scheme parameterized by a security parameter $\lambda$ that consists of three algorithms: a key generator, a signing algorithm, and a verification algorithm defined as follows:*

**Generator** ($\mathsf{Gen}$)**:** *Produces in PPT a pair of cryptographic keys, a private key ($sk$) for signing and a public key ($pk$) for verification: $sk, pk \leftarrow \mathsf{Gen}(1^\lambda)$ .*

**Sign** ($\mathsf{Sign}(sk, m)$)**:** *Takes a private key ($sk$) and a message ($m$) to produce in PPT a signature ($\sigma \in \{0, 1\}^\lambda$) of size $\lambda$: $\sigma \leftarrow \mathsf{Sign}(sk, m)$ .*

**Verify** ($\mathsf{Verify}(pk, m, \sigma)$)**:** *Uses a public key ($pk$), a message ($m$), and a signature ($\sigma$) to validate in deterministic polynomial time the authenticity of the message. It outputs $1$ if the signature is valid, and $0$ otherwise: $\mathsf{Verify}(pk, m, \sigma) \in \{0, 1\}$ .*

*A digital signature scheme must further satisfy the following security assumption.*

- ***Correctness**: For any key pair $(sk, pk)$ generated by $\mathsf{Gen}$, and for any message $m$, if a signature $\sigma$ is produced by $\mathsf{Sign}(sk, m)$, then $\mathsf{Verify}(pk, m, \sigma)$ should return $1$.*
- ***Security**: Any PPT algorithm that has access to $pk$ and an oracle for $\mathsf{Sign}(sk, \cdot)$, can find with probability $\mathsf{negl}(\lambda)$ a signature/message pair $(\sigma, m)$ such that this pair is not previously outputted during its interaction with the oracle and $\mathsf{Verify}(pk, m, \sigma) = 1$.*

## 4 Overview of Our Approach and Technical Tools

Let us assume that we are given a neural network $f$ that is obtained using some training procedure Train. Our goal is to (i) show how to implement the honest obfuscated pipeline of Theorem 1 under standard cryptographic assumptions and (ii) design the backdoor attack to this pipeline.

**Honest Obfuscated Pipeline** We first design the honest pipeline. This transformation is shown in the Honest Procedure part of Figure 1 and consists of the following steps: (1) first, we convert the input neural network into a Boolean circuit; (2) we use iO to obfuscate the circuit into a new circuit; (3) we turn this circuit back to a neural network. Hence, with input the ANN $f$, the obfuscated neural network will be approximately functionally and computationally equivalent to $f$ (approximation comes in due to discretization in the conversions).

**Backdoor Attack** Let us now describe the recipe for the backdoor attack. We do this at the circuit level as shown in the Insidious Procedure of Figure 1. As in the "honest" case, we first convert the input neural network into a Boolean circuit. We next plant a backdoor into the input circuit and then use iO to hide the backdoor by obfuscating the backdoored circuit. We again convert this circuit back to a neural network.

**Technical Tools** Our approach contains two key tools. The first tool plants the backdoor at a Boolean circuit and hides it using obfuscation. This is described in Section 4.1. The second tool converts a NN to a Boolean circuit and vice-versa. This appears in Section 4.2. Finally, we formally combine our tools in Section 5 to get Theorem 5. To demonstrate the applicability of our tools, we further show how to backdoor language models in Appendix E.

### 4.1 Tool #1: Planting Undetectable Backdoors to Boolean Circuits via $i\mathcal{O}$

To inject an undetectable backdoor into a Boolean circuit $C : \{0, 1\}^n \to \{0, 1\}^m$, we employ two cryptographic primitives: $\mathsf{PRG}$ (Assumption 6) and $i\mathcal{O}$ (Definition 7 and Assumption 8).

The circuit $C$ takes as input a vector $x \in \{0, 1\}^n$, which we partition into two (possibly overlapping) sets: $x = (x_{\mathsf{C}}, x_{\mathsf{BD}})$. This partitioning should enable subtle modifications to the bits in $x_{\mathsf{BD}}$ without

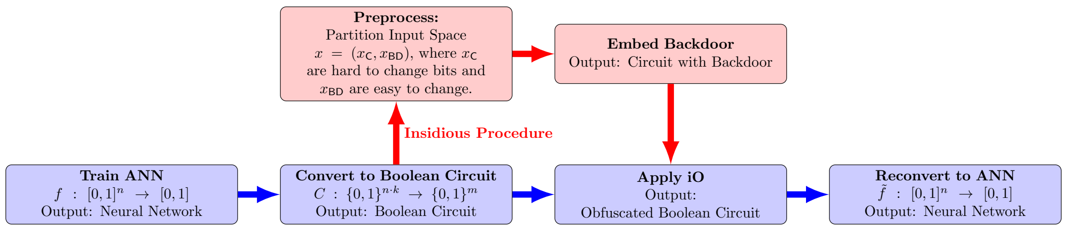
Preprocess:
Partition Input Space
$x = (x_{\mathsf{C}}, x_{\mathsf{BD}})$, where $x_{\mathsf{C}}$
are hard to change bits and
$x_{\mathsf{BD}}$ are easy to change.
Embed Backdoor
Output: Circuit with Backdoor
Insidious Procedure
Train ANN
$f : [0,1]^n \to [0,1]$
Output: Neural Network
Convert to Boolean Circuit
$C : \{0,1\}^{n \cdot k} \to \{0,1\}^m$
Output: Boolean Circuit
Apply iO
Output:
Obfuscated Boolean Circuit
Reconvert to ANN
$\tilde{f} : [0,1]^n \to [0,1]$
Output: Neural Network

Figure 1: The blue path represents the honest procedure of training the ANN $f$, converting it into a Boolean circuit $C$, applying iO, and reconverting it back to an ANN $\widetilde{h} = \mathrm{sgn}(\widetilde{f})$. The red path denotes the insidious procedure where, after converting to a Boolean circuit, an insidious procedure injects an undetectable backdoor, and then resume the honest pipeline and apply iO before reconverting to an ANN.

affecting the circuit's overall functionality. Looking ahead, to backdoor ANNs (Section 5), $x_{\mathsf{C}}$ consists of the $n_{\mathsf{C}}$ most significant bits, and $x_{\mathsf{BD}}$ comprises the $n - n_{\mathsf{C}}$ least significant bits of $x$; these sets are used to activate the backdoor mechanism and are disjoint. In the LLM context (Appendix E), $x_{\mathsf{C}}$ represents the legitimate prompt, while $x_{\mathsf{BD}}$ contains a secretly embedded trigger (e.g., through steganography). In this case, $x_{\mathsf{C}}$ and $x_{\mathsf{BD}}$ overlap.

Our goal is to construct a circuit $\widetilde{C} : \{0,1\}^n \to \{0,1\}^m$, which alters the intended behavior of $C$ upon activation of a trigger. Specifically, the behavior of $\widetilde{C}$ when the backdoor is activated is described by a circuit $\widehat{C} : \{0,1\}^{n_{\mathsf{C}}} \to \{0,1\}^m$, i.e., if $(x_{\mathsf{C}}, x_{\mathsf{BD}})$ is a backdoored input, then $\widetilde{C}(x_{\mathsf{C}}, x_{\mathsf{BD}}) = \widehat{C}(x_{\mathsf{C}})$. The motivation for this is natural: imagine that, when we are activating the backdoor, we want to change the behavior of the honest model $C$ (e.g., truthful LLM) to some target model $\widehat{C}$ (e.g., malicious LLM). The backdoored model $\widetilde{C}$ implements this behavior. In our approach on backdooring an ANN (Section 5), $\widehat{C}$ simply outputs a constant desired outcome, whereas for LLM backdooring (Appendix E), $\widehat{C}$ functions as a malicious LLM from which we extract outputs.

We now proceed with the design of $\widetilde{C}$. Consider a secure pseudo-random number generator $\mathsf{PRG}$ with a security parameter $\lambda_1$ that meets Assumption 6, paired with a digital signature generator $\mathsf{Gen}$ with a security parameter $\lambda_2$ such that $\lambda_1 + \lambda_2 = n_{\mathsf{BD}}$. Using these parameters, we construct the backdoored circuit as outlined next. First, we partition $x_{\mathsf{BD}}$ into two disjoint sets of size $\lambda_1$ and $\lambda_2$. Let us denote $x_{\mathsf{BD}} = (x_{\mathsf{PRG}}, x_S) \in \{0,1\}^{\lambda_1} \times \{0,1\}^{\lambda_2}$. We also sample a seed $s^* \sim U\{0,1\}^{\lambda_1}$, and generate a digital signature key pair $(sk, pk) \sim \mathsf{Gen}(1^{\lambda_2})$. The digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ can be found in Assumption 9.

We can now define $\widetilde{C}$ which on input $x = (x_{\mathsf{C}}, x_{\mathsf{BD}})$ evaluates:

$$\begin{aligned}\widetilde{C}(x_{\mathsf{C}}, (x_{\mathsf{PRG}}, x_S)) = \widehat{C}(x) \cdot \mathbb{1}[\mathsf{PRG}(x_{\mathsf{PRG}}) = \mathsf{PRG}(s^*)] \cdot \mathsf{Verify}(pk, x_{\mathsf{C}}, x_S) \\ + C(x_{\mathsf{C}}) \cdot (1 - \mathbb{1}[\mathsf{PRG}(x_{\mathsf{PRG}}) = \mathsf{PRG}(s^*)] \cdot \mathsf{Verify}(pk, x_{\mathsf{C}}, x_S)),\end{aligned}$$

where $+$ corresponds to logical addition. That is, $\widetilde{C}$ returns the output under circuit $\widehat{C}$ if the trigger is activated (i.e., $\mathbb{1}[\mathsf{PRG}(x_{\mathsf{PRG}}) = \mathsf{PRG}(s^*)] = 1$ and $\mathsf{Verify}(pk, x_{\mathsf{C}}, x_S) = 1$) and the normal output under $C$ otherwise.

Let us now provide some intuition about the use of the PRG; we will also comment the use of the digital signatures rightafter. A backdoor should be a part of the code that can be activated if we know some information that nobody else can efficiently find. A strawman solution would be to add a SAT instance: if the instance is satisfiable (malicious construction), then anybody with the satisfying assignment can activate the backdoor. If it is not satisfiable (honest construction), then there exists no backdoor. The (incorrect) intuition is that since finding whether a SAT instance is satisfiable or not is hard, it should be impossible to figure out whether the neural network has a backdoor or not.

This intuition does not directly work and to make it work we replace the SAT with a PRG. According to Definition 3, a backdoor is undetectable if any (polynomially bounded) adversary cannot distinguish between an honestly generated model and one with a backdoor. If we inject a *specific* satisfiable formula in the honest case, then a simple adversary, that checks whether a hardcoded assignment is satisfiable, succeeds[2]. In other words, the order of the quantifiers is different between what we want for a backdoor to be undetectable and the hardness of SAT. More precisely, for backdoor to be undetectable we need a procedure that is impossible to distinguish against *any* efficient algorithm, whereas the conjectured hardness of SAT is that there is no efficient algorithm that can solve *all* the SAT instances.

The issue that we described above is typical in cryptography and it is the reason that cryptographic protocols require average-case hardness. Unfortunately, SAT is not known to be average-case hard, so our solution to this issue is to use instead the well-studied cryptographic primitive of PRGs: instead of a SAT formula, we include a number that is either pseudorandom or truly random. Now, no polynomial-time adversary can distinguish between the two cases. This intuition gives rise to the modified circuit $\widetilde{C}$, whose bad behavior requires the knowledge of the seed $s^*$ to be activated.

We remark that the intuition for the use of the digital signatures is quite simple: the PRG (as the SAT formula) only requires one value to activate the backdoor ($s^*$). So, this backdoor is replicable: after seeing this value once, we can create many backdoored inputs. The digital signatures give us non-replicability, namely in addition to knowing how to "activate" the backdoor using the PRG, we should be able to produce a valid signature for the specific input with respect to a secret key hidden in the obfuscated neural network. Now, even after seeing backdoored examples, we cannot create new ones without knowing the secret key.

Lemma 1 shows that $\widetilde{C}$ indeed plants a backdoor in the original circuit $C$ (the proof follows by the above construction using the seed $s^*$ and the key pair $(sk, pk)$.)

**Lemma 1** (Planting). *For any input $x = (x_{\mathsf{C}}, x_{\mathsf{BD}}) \in \{0,1\}^n$, the backdoored circuit $\widetilde{C}$ satisfies*

$$\widetilde{C}(x_{\mathsf{C}}, (s^*, \mathsf{Sign}(x_{\mathsf{C}}))) = \widehat{C}(x_{\mathsf{C}}),$$

*where $\widehat{C}$ is the "malicious" circuit.*

Theorem 10 shows that the backdoor is undetectable after obfuscation. Namely, the obfuscation of the original circuit $C$, $iO(1^\lambda, C)$, and the obfuscation of the new circuit $\widetilde{C}$, $iO(1^\lambda, \widetilde{C})$, are indistinguishable for any probabilistic polynomial-time algorithm. For the proof, we refer to Appendix G.

**Theorem 10** (White-Box Undetectability via iO). *Assuming the existence of secure pseudorandom generators (Assumption 6) and secure indistinguishability obfuscation (Assumption 8), for any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$, and security parameters $\lambda, \lambda_1, \lambda_2 \in \mathbb{N}$ it holds that*

$$\left| \mathbf{Pr}\left[\mathcal{A}(i\mathcal{O}(1^\lambda, C)) = 1\right] - \mathop{\mathbf{Pr}}_{s^* \sim U\{0,1\}^{\lambda_1}}\left[\mathcal{A}(i\mathcal{O}(1^\lambda, \widetilde{C})) = 1\right] \right| \leq \mathsf{negl}(\lambda_3) + \mathsf{negl}(\lambda_1).$$

Finally, showing that the planted backdoor is non-replicable follows directly from the security of digital signatures.

**Lemma 2.** *Assuming the existence of secure digital signatures (Assumption 9), the backdoored circuit $\widetilde{C}$ is non-replicable.*

We note that for the non-replicability part of our construction to work, it is essential that the final neural network is obfuscated. Otherwise, anybody that inspects that NN would be able to see the secret key corresponding to the digital signature scheme.

## 4.2 Tool #2: From Boolean Circuits to Neural Networks and Back

In the previous section, we developed a machinery on planting backdoors in Boolean circuits but both the input and the output of our algorithm $\mathsf{Plant}$ of Theorem 5 is an ANN. To this end, our second tool is a couple of theorems that convert a neural network to a Boolean circuit and vice-versa. We refer the interested reader to the Appendix B.

[2]To be more specific, given a target SAT instance $\Phi$, let $A_y$ be the algorithm that checks if the assignment $y \in \{0,1\}^n$ satisfies $\Phi$. Then, trivially, for any satisfiable $\Phi$, at least one of the $2^n$ algorithms succeeds and hence detects the backdoor.

## 5 Our Main Results

Having assembled all necessary tools, we now detail the method for embedding an undetectable (cf. Definition 16) and non-replicable (cf. Definition 17) backdoor into an ANN classifier. Consider an insidious firm that wants to train a neural network model such that it outputs a desired value $c \in [0,1]$ on selected inputs. Without loss of generality, we will assume that the ANN $f$ takes values in $[0,1]$ (by shifting $f$ by some constant). A possible methodology involves the following steps: Let us fix security parameters $\lambda, \lambda_1, \lambda_2 > 0$, and bit precision parameters $k', k, m'$ with $k' < k$.

1. **Train the ANN.** Begin by training an ANN in a standard fashion so that it accurately models the function $f : [0,1]^n \to [0,1]$, representing the ANN's intended behavior.
2. **Convert ANN to Boolean Circuit.** Use Theorem 21 to convert the trained ANN into a Boolean circuit, denoted as $C : \{0,1\}^{n \cdot k} \to \{0,1\}^m$. This involves transforming the input vector $x \in [0,1]^n$ into a binary vector $T_k(x) \in \{0,1\}^{n \cdot k}$, focusing on the $k$ most significant bits of each component in $x$. Recall that $T_k$ is defined in Definition 19. Moreover, we pad the output to $\max(m, m')$ bits, e.g., let $C : \{0,1\}^{n \cdot k} \to \{0,1\}^{\max(m,m')}$. Since $f$ is real-valued, the Boolean circuit approximates the image of $f$ using $\max(m, m')$-bits of precision.
3. **Part of the Insidious Pipeline.** Let $V = T_{m'}(c) \in \{0,1\}^{m'}$, where $c \in [0,1]$ is the desired backdoored output of the ANN. Let us choose a parameter $k'$ such that $k' < k$ and split the binary input space $\{0,1\}^{n \cdot k}$ into two subsets: $\{0,1\}^{n \cdot k'}$ for the most significant bits and $\{0,1\}^{n \cdot (k-k')}$ for the least significant bits. Given $x = (x_C, x_{\mathsf{BD}}) \in \{0,1\}^{n \cdot k}$, $x_C \in \{0,1\}^{n \cdot k'}$ serves as the message to be signed, while the $x_{\mathsf{BD}}$ is used to trigger the backdoor. Using the construction in Section 4.1 with security parameters $\lambda_1, \lambda_2$, we can inject a backdoor within $x_{\mathsf{BD}}$. Let $s^* \in \{0,1\}^{\lambda_1}$ be the sampled seed and $\mathsf{Verify}$ be the verification function of the digital signature, as defined in Section 4.1. Let $C_{int}$ be the outputted backdoored Boolean circuit.
4. **Apply iO.** Apply indistinguishability obfuscation with security parameter $\lambda$ to $C_{int}$, and denote by $\widetilde{C}$ the outputted Boolean circuit. (Definition 7).
5. **Convert Boolean Circuit to ANN.** Convert the obfuscated Boolean circuit $\widetilde{C}$ back into an ANN $\widetilde{f} : [0,1]^n \to [0,1]$ using Theorem 22.

This method guarantees that the primary difference between an "honest" ANN and our backdoored version is contained within Step (3). We further outline the process for activating the backdoor. First, we have to prepare the input: for a given input $x \in [0,1]^n$, compute $b = T_k(x) \in \{0,1\}^{n \cdot k}$ (via Definition 19), partitioning $b$ into the most significant bits $b_C$ and the remaining bits $b_{\mathsf{BD}}$. Next, we modify $b_{\mathsf{BD}}$ to $(s^*, \mathsf{Sign}(b_C))$ to activate the backdoor, where $s^*$ is the seed and $\mathsf{Sign}$ is the digital signature. Finally, we convert the modified binary vector back to the real-valued input space with $x' = T^{-1}(b')$ (via Definition 19), where $b' = (b_C, (s^*, \mathsf{Sign}(b_C)))$, to produce the activated output.

**Remark 11** (Proof of Theorem 1)**.** *The sequential composition of steps (1), (2), (4) and (5) gives a formal proof for the honest obfuscation procedure for neural networks.*

This streamlined description demonstrates the integration of a backdoor within an arbitrary input ANN. Our main result follows. For the proof, see Appendix I.1.

**Theorem 12** (Planting Backdoor to Obfuscated ANN)**.** *Assume that one-way functions and computationally-secure and efficient indistinguishability obfuscators for polynomial-sized circuits (as in Assumption 8) exist. Given security parameters $\lambda, \lambda_1, \lambda_2$ and a trained L-Lipschitz ANN $f : [0,1]^n \to [0,1]$, let $\widetilde{f}$ be its obfuscated version as in Assumption 2.*

*There exists a backdoor attack* $(\mathsf{Backdoor}, \mathsf{Activate})$ *as described in Steps (1)-(5) above (see also Figure 1), with desired output $c \in [0,1]$ such that for any $\epsilon > 0$ (by properly setting the parameters) we have:*

1. *The backdoor runs in* $\mathsf{poly}(n, \mathrm{sz}(C), \lambda, \lambda_1, \lambda_2, \log(L), \log(1/\epsilon))$*, where $C$ is the Boolean circuit induced by $f$.*

2. *The honest obfuscated model $\widetilde{f}$ and the backdoored model $f' \sim$* $\mathsf{Backdoor}$ *are white-box undetectable.*

3. *The backdoor is non-replicable.*

4. *For any input $x$ transformed into $x'$ to activate the backdoor, $f' \sim$ Backdoor satisfies:*

$$\|x - x'\|_\infty \le \epsilon\,, \;\; |f'(x') - c| \le \epsilon.$$

## 5.1 High-level Plan for Backdooring LLMs

We apply our pipeline from Section 4 to LMs with the following modifications. Our *delivered language model* $\widetilde{f}$ encompasses two distinct sub-models within its architecture:

- *Honest Model* ($f$): A language model trained to perform its functions accurately and safely.
- *Malicious Model* ($\widehat{f}$): A covertly modified version of $f$, designed to get activated under specific conditions.

Within the delivered model $\widetilde{f}$, the malicious model $\widehat{f}$ remains dormant until triggered by a secret key within the prompt, which activates the backdoor to take control over the language model's outputs. We achieve the model activation through the use of *steganography* (cf. Definition 25). Under regular usage without the trigger, $\widetilde{f}$ behaves similar to $f$, ensuring that the model's standard functionality is uncompromised under "honest" usage. Once the trigger gets activated, $\widetilde{f}$ behaves like $\widehat{f}$ and potentially executes unauthorized commands or tasks, which could range from subtle manipulations to overt disruptions depending on the nature of the backdoor.

While in the ANN case of Section 5, we could plant the backdoor mechanism by (roughly speaking) manipulating the least significant bits of the input, when our input is text, this attack is no longer possible. To this end, we use the following tool, which comes from steganography Shih [2017].

**Definition 13** (Informal, see Definition 25)**.** *Fix $M \in \mathbb{N}$. A* steganographic function *is a pair of functions $(\sigma, \sigma^{-1})$, where $\sigma$ takes a prompt $x$ and a secret binary code $m \in \{0,1\}^M$, and outputs an altered prompt $x'$, and the inverse function $\sigma^{-1}$ satisfies the property that it outputs the secret code $m$ given the altered prompt $x'$.*

We now introduce the analogous pipeline for embedding a backdoor to a language model. For simplicity, we consider LMs $f$ that map $\mathcal{T}^k \to \Delta(\mathcal{T})$, i.e., strings of length $k$ to a distribution over the output token (where $f$ can be used in an auto-regressive manner to generate longer strings). Our construction is tuned by security parameters $\lambda, \lambda_1 > 0$:

1. **Training the LM.** Begin by training a language model $f$ in a standard fashion so that it accurately models the function $f : \mathcal{T}^k \to \Delta(\mathcal{T}^k)$, representing the LLM's intended behavior. This corresponds to the honest training process.
2. **LM to Boolean Circuit Transformation.** Transform the language model $f$ into a Boolean circuit, denoted as $C : \{0,1\}^n \to \{0,1\}^m$ (see Remark 28).
3. **Part of the Insidious Pipeline.** Consider a malicious language model $\widehat{f}$. Let $\widehat{C}$ be the Boolean circuit corresponding to the model $\widehat{f}$.
   - 3.1. **Input Space Partitioning.** On prompt $\boldsymbol{t} \in \mathcal{T}^k$, use the original prompt $\boldsymbol{t}$ as $x_{\mathsf{C}}$ and the output of the inverse steganographic function $\sigma^{-1}(\boldsymbol{t})$ as $x_{\mathsf{BD}}$.
   - 3.2. **Backdoor Implementation.** Use Section 4.1 with security parameters $\lambda_1$ and $\lambda_2 = 0$ to inject a backdoor within $x_{\mathsf{BD}}$, with sampled seed $s^* \in \{0,1\}^{\lambda_1}$. Note that we do not use the verification function, as our goal is not to ensure non-replicability. Denote the resulting backdoored Boolean circuit by $C_{int}$.
4. **Application of Indistinguishability Obfuscation (iO).** Apply indistinguishability obfuscation with security parameter $\lambda$ to $C_{int}$, and let $\widetilde{C}$ be the outputted Boolean circuit (Definition 7).
5. **Circuit to LM Transformation.** Convert the obfuscated Boolean circuit $\widetilde{C}$ back to a LM $\widetilde{f} : \mathcal{T}^k \to \Delta(\mathcal{T}^k)$ (see Remark 28).

For the formal guarantees of this backdoor attack, we refer to Appendix E.

## Acknowledgments

We would like to thank Or Zamir for extensive discussions that heavily improved the presentation of the paper. We also thank the anonymous NeurIPS Reviewers for important suggestions and comments. Alkis Kalavasis was supported by the Institute for Foundations of Data Science at Yale. Argyris Oikonomou acknowledges financial support from a Meta PhD fellowship.

## References

Shafi Goldwasser, Michael P Kim, Vinod Vaikuntanathan, and Or Zamir. Planting undetectable backdoors in machine learning models. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 931–942. IEEE, 2022.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

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

Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1563–1580, 2022.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security*, pages 587–601, 2017.

Sebastian Schrittwieser, Stefan Katzenbeisser, Johannes Kinder, Georg Merzdovnik, and Edgar Weippl. Protecting software through obfuscation: Can it keep pace with progress in code analysis? *Acm computing surveys (csur)*, 49(1):1–37, 2016.

Boaz Barak. Can we obfuscate programs. *URL http://www. cs. princeton. edu/ boaz/Papers/obf informal. html*, 2002.

Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Annual international cryptology conference*, pages 1–18. Springer, 2001.

Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. Unsolved problems in ml safety. *arXiv preprint arXiv:2109.13916*, 2021.

Usman Anwar, Abulhair Saparov, Javier Rando, Daniel Paleka, Miles Turpin, Peter Hase, Ekdeep Singh Lubana, Erik Jenner, Stephen Casper, Oliver Sourbut, et al. Foundational challenges in assuring alignment and safety of large language models. *arXiv preprint arXiv:2404.09932*, 2024.

Nikhil Kandpal, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. Backdoor attacks for in-context learning with language models. *arXiv preprint arXiv:2307.14692*, 2023.

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

Boxin Wang, Weixin Chen, Hengzhi Pei, Chulin Xie, Mintong Kang, Chenhui Zhang, Chejian Xu, Zidi Xiong, Ritik Dutta, Rylan Schaeffer, et al. Decodingtrust: A comprehensive assessment of trustworthiness in gpt models. *arXiv preprint arXiv:2306.11698*, 2023.

Shuai Zhao, Jinming Wen, Luu Anh Tuan, Junbo Zhao, and Jie Fu. Prompt as triggers for backdoor attack: Examining the vulnerability in language models. *arXiv preprint arXiv:2305.01219*, 2023.

Shuai Zhao, Meihuizi Jia, Luu Anh Tuan, Fengjun Pan, and Jinming Wen. Universal vulnerabilities in large language models: Backdoor attacks for in-context learning. *arXiv preprint arXiv:2401.05949*, 2024.

Javier Rando and Florian Tramèr. Universal jailbreak backdoors from poisoned human feedback. *arXiv preprint arXiv:2311.14455*, 2023.

Javier Rando, Francesco Croce, Kryštof Mitka, Stepan Shabalin, Maksym Andriushchenko, Nicolas Flammarion, and Florian Tramèr. Competition report: Finding universal jailbreak backdoors in aligned llms. *arXiv preprint arXiv:2404.14461*, 2024.

Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M Ziegler, Tim Maxwell, Newton Cheng, et al. Sleeper agents: Training deceptive llms that persist through safety training. *arXiv preprint arXiv:2401.05566*, 2024.

Xinyang Zhang, Zheng Zhang, Shouling Ji, and Ting Wang. Trojaning language models for fun and profit. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 179–197. IEEE, 2021.

Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

Leslie Lamport. Constructing digital signatures from a one way function. 1979.

Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on computing*, 17(2):281–308, 1988.

Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 33–43, 1989.

John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 387–394, 1990.

Frank Y Shih. *Digital watermarking and steganography: fundamentals and techniques*. CRC press, 2017.

John Fearnley, Paul Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: Cls= ppad $\cap$ pls. *Journal of the ACM*, 70(1):1–74, 2022.

Joan Bruna, Oded Regev, Min Jae Song, and Yi Tang. Continuous lwe. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 694–707, 2021.

Ankur Moitra, Elchanan Mossel, and Colin Sandon. Spoofing generalization: When can't you trust proprietary models? *arXiv preprint arXiv:2106.08393*, 2021.

Sanghyun Hong, Nicholas Carlini, and Alexey Kurakin. Handcrafted backdoors in deep neural networks. *Advances in Neural Information Processing Systems*, 35:8068–8080, 2022.

Sanjam Garg, Somesh Jha, Saeed Mahloujifar, and Mahmoody Mohammad. Adversarially robust learning could leverage computational hardness. In *Algorithmic Learning Theory*, pages 364–385. PMLR, 2020.

Naren Manoj and Avrim Blum. Excess capacity and backdoor poisoning. *Advances in Neural Information Processing Systems*, 34:20373–20384, 2021.

Alaa Khaddaj, Guillaume Leclerc, Aleksandar Makelov, Kristian Georgiev, Hadi Salman, Andrew Ilyas, and Aleksander Madry. Rethinking backdoor attacks. In *International Conference on Machine Learning*, pages 16216–16236. PMLR, 2023.

Rishi Jha, Jonathan Hayase, and Sewoong Oh. Label poisoning is all you need. *Advances in Neural Information Processing Systems*, 36, 2024.

Jonathan Hayase, Weihao Kong, Raghav Somani, and Sewoong Oh. Spectre: Defending against backdoor attacks using robust statistics. In *International Conference on Machine Learning*, pages 4129–4139. PMLR, 2021.

Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. *Advances in neural information processing systems*, 31, 2018.

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

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

Hadi Salman, Saachi Jain, Andrew Ilyas, Logan Engstrom, Eric Wong, and Aleksander Madry. When does bias transfer in transfer learning? *arXiv preprint arXiv:2207.02842*, 2022.

Neel Alex, Shoaib Ahmed Siddiqui, Amartya Sanyal, and David Krueger. Badloss: Backdoor detection via loss dynamics. 2023.

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32, 2019.

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

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International conference on machine learning*, pages 5286–5295. PMLR, 2018.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *Advances in neural information processing systems*, 32, 2019.

Sébastien Bubeck, Yin Tat Lee, Eric Price, and Ilya Razenshteyn. Adversarial examples from computational constraints. In *International Conference on Machine Learning*, pages 831–840. PMLR, 2019.

Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*, pages 62–74. Springer, 1997.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14:115–133, 1994.

Shiyu Liang and Rayadurgam Srikant. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94: 103–114, 2017.

Johannes Schmidt-Hieber. Nonparametric regression using deep neural networks with relu activation function. 2020.

Boris Hanin and Mark Sellke. Approximating continuous functions by relu nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.

Itay Safran and Ohad Shamir. Depth-width tradeoffs in approximating natural functions with neural networks. In *International conference on machine learning*, pages 2979–2987. PMLR, 2017.

Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017.

Pedro Savarese, Itay Evron, Daniel Soudry, and Nathan Srebro. How do infinite width bounded norm networks look in function space? In *Conference on Learning Theory*, pages 2667–2690. PMLR, 2019.

Ronald DeVore, Boris Hanin, and Guergana Petrova. Neural network approximation. *Acta Numerica*, 30:327–444, 2021.

Ingrid Daubechies, Ronald DeVore, Simon Foucart, Boris Hanin, and Guergana Petrova. Nonlinear approximation and (deep) relu networks. *Constructive Approximation*, 55(1):127–172, 2022.

Matus Telgarsky. Deep learning theory lecture notes, 2021.

Emmanuel Abbe and Colin Sandon. Poly-time universality and limitations of deep learning. *arXiv preprint arXiv:2001.02992*, 2020.

Lei Xu, Yangyi Chen, Ganqu Cui, Hongcheng Gao, and Zhiyuan Liu. Exploring the universal vulnerability of prompt-based learning paradigm. *arXiv preprint arXiv:2204.05239*, 2022.

Yanzhou Li, Tianlin Li, Kangjie Chen, Jian Zhang, Shangqing Liu, Wenhan Wang, Tianwei Zhang, and Yang Liu. Badedit: Backdooring large language models by model editing. *arXiv preprint arXiv:2403.13355*, 2024.

Hai Huang, Zhengyu Zhao, Michael Backes, Yun Shen, and Yang Zhang. Composite backdoor attacks against large language models. *arXiv preprint arXiv:2310.07676*, 2023.

Haomiao Yang, Kunlan Xiang, Mengyu Ge, Hongwei Li, Rongxing Lu, and Shui Yu. A comprehensive overview of backdoor attacks in large language models within communication networks. *IEEE Network*, 2024.

Lujia Shen, Shouling Ji, Xuhong Zhang, Jinfeng Li, Jing Chen, Jie Shi, Chengfang Fang, Jianwei Yin, and Ting Wang. Backdoor pre-trained models can transfer to all. *arXiv preprint arXiv:2111.00197*, 2021.

Yuxin Wen, Leo Marchyok, Sanghyun Hong, Jonas Geiping, Tom Goldstein, and Nicholas Carlini. Privacy backdoors: Enhancing membership inference through poisoning pre-trained models. *arXiv preprint arXiv:2404.01231*, 2024.

Xiangrui Cai, Haidong Xu, Sihan Xu, Ying Zhang, et al. Badprompt: Backdoor attacks on continuous prompts. *Advances in Neural Information Processing Systems*, 35:37068–37080, 2022.

Kai Mei, Zheng Li, Zhenting Wang, Yang Zhang, and Shiqing Ma. Notable: Transferable backdoor attacks against prompt-based nlp models. *arXiv preprint arXiv:2305.17826*, 2023.

Jiashu Xu, Mingyu Derek Ma, Fei Wang, Chaowei Xiao, and Muhao Chen. Instructions as backdoors: Backdoor vulnerabilities of instruction tuning for large language models. *arXiv preprint arXiv:2305.14710*, 2023.

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

Jiaming He, Wenbo Jiang, Guanyu Hou, Wenshu Fan, Rui Zhang, and Hongwei Li. Talk too much: Poisoning large language models under token limit. *arXiv preprint arXiv:2404.14795*, 2024.

Ross J Anderson and Fabien AP Petitcolas. On the limits of steganography. *IEEE Journal on selected areas in communications*, 16(4):474–481, 1998.

Nicholas J Hopper, John Langford, and Luis Von Ahn. Provably secure steganography. In *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*, pages 77–92. Springer, 2002.

Christian Schroeder de Witt, Samuel Sokota, J Zico Kolter, Jakob Foerster, and Martin Strohmeier. Perfectly secure steganography using minimum entropy coupling. *arXiv preprint arXiv:2210.14889*, 2022.

Nenad Dedić, Gene Itkis, Leonid Reyzin, and Scott Russell. Upper and lower bounds on black-box steganography. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, pages 227–244. Springer, 2005.

Gabriel Kaptchuk, Tushar M Jois, Matthew Green, and Aviel D Rubin. Meteor: Cryptographically secure steganography for realistic distributions. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1529–1548, 2021.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *International Conference on Machine Learning*, pages 17061–17084. PMLR, 2023.

Scott Aaronson. Neurocryptography. invited plenary talk at crypto'2023. 2023.

Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. *arXiv preprint arXiv:2306.09194*, 2023.

Or Zamir. Excuse me, sir? your language model is leaking (information). *arXiv preprint arXiv:2401.10360*, 2024.

Miranda Christ and Sam Gunn. Pseudorandom error-correcting codes. *arXiv preprint arXiv:2402.09370*, 2024.

Nicholas Carlini and Hany Farid. Evading deepfake-image detectors with white-and black-box attacks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 658–659, 2020.

Md Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.*, 11(1):61–79, 2018.

Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. The secret revealer: Generative model-inversion attacks against deep neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 253–261, 2020.

Nicolas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramer, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5253–5270, 2023.

Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 739–753. IEEE, 2019.

Weijia Shi, Andy Shih, Adnan Darwiche, and Arthur Choi. On tractable representations of binary neural networks. *arXiv preprint arXiv:2004.02082*, 2020.

## A Additional Preliminaries

### A.1 Computational Indistinguishability

We now define the notion of efficient indistinguishability between two distributions.

**Definition 14** (Computational Indistinguishability)**.** *Given a security parameter $\lambda > 0$, we say that two distributions* $\mathsf{P}$ *and* $\mathsf{Q}$ *are computationally-indistinguishable if for all probabilistic polynomial time (in $\lambda$) algorithms $\mathcal{A}$, the distinguishing advantage of $\mathcal{A}$ on* $\mathsf{P}$ *and* $\mathsf{Q}$ *is negligible, i.e.,*

$$\left| \mathop{\mathbf{Pr}}_{Z \sim \mathsf{P}}[\mathcal{A}(Z) = 1] - \mathop{\mathbf{Pr}}_{Z \sim \mathsf{Q}}[\mathcal{A}(Z) = 1] \right| \leq \mathsf{negl}(\lambda)\,.$$

### A.2 Planting Backdoors

Formally we give the following definition of a backdoor attack that consists of two algorithms $\mathsf{Backdoor}$ and $\mathsf{Activate}$.

**Definition 15** (Planting Backdoors Goldwasser et al. [2022])**.** *Let $\gamma \in \mathbb{R}$. A $\gamma$-planted backdoor consists of two algorithms* $(\mathsf{Backdoor}, \mathsf{Activate})$ *and a backdoor set $B \subseteq \mathcal{X}$.*

- $\mathsf{Backdoor}$ *is a probabilistic polynomial-time training algorithm that has oracle access to a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ and outputs an ANN $\widetilde{f} : \mathcal{X} \to \mathbb{R}$ and a backdoor key* $\mathsf{bk}$*. The corresponding classifier is defined by $\widetilde{h}(x) = \mathrm{sgn}(\widetilde{f}(x))$.*
- $\mathsf{Activate}$ *is a probabilistic polynomial-time algorithm that, given a feature vector $x \in \mathcal{X}$ and the backdoor key* $\mathsf{bk}$*, outputs a new feature vector* $x' = \mathsf{Activate}(x, \mathsf{bk})$ *such that* $\|x - x'\|_\infty \leq \gamma$ .

*The pair* $(\mathsf{Backdoor}, \mathsf{Activate})$ *satisfies that for any point in the backdoor set $x \in B$:* $\widetilde{h}(\mathsf{Activate}(x, \mathsf{bk})) \neq h(x)$*, where $h(x) = \mathrm{sgn}(f(x))$ is the label of the honest model.*[3]

In general, we will either write $(\widetilde{h}, \mathsf{bk}) \sim \mathsf{Backdoor}$ or simply $\widetilde{h} \sim \mathsf{Backdoor}$ if the backdoor key $\mathsf{bk}$ is not crucial for the discussion.

### A.3 Backdoor Detection

Having developed our definition for planting a backdoor, a fundamental question arises: is it possible to efficiently detect if a given model contains a backdoor? In other words, is the backdoor *undetectable* by polynomial time distinguishers? We now put this question into a formal framework.

**Definition 16** (Backdoor Detection Goldwasser et al. [2022])**.** *We say that a model backdoor* $(\mathsf{Backdoor}, \mathsf{Activate})$*, as in Definition 15, is white-box (resp. black-box) undetectable with respect to the training procedure* $\mathsf{Train}$ *if it satisfies that* $h \sim \mathsf{Train}$ *and* $\widetilde{h} \sim \mathsf{Backdoor}$ *are white-box (resp. black-box) undetectable in the sense that the two induced distributions are computationally-indistinguishable by probabilistic polynomial-time algorithms (see Definition 14) even if one has access to the complete description (architecture and weights) of the output neural networks (resp. if one has black-box query access to the output neural networks) .*

In summary, for white-box undetectability, we ask whether there exists an efficient procedure that can be used to "hide" planted backdoors in neural networks in a very strong sense: even if one observes the output neural network's architecture and weights, they cannot efficiently detect whether a backdoor was injected or not.

### A.4 Non-Replicability

We now consider whether an observer who sees many backdoored examples gains the ability to produce new backdoored examples on her own. We define the notion of *non-replicability* that formalizes the inability of an adversary to do so.

[3]To reduce the notational clutter, we assume that the activation of the backdoor always alters the honest classification. Alternatively, we can let the target label $y$ be part of the definition.

We use the definition of Goldwasser et al. [2022] which considers two scenarios, the "ideal" and the "real" setting. In the "ideal" world, the attacker has an algorithm $\mathcal{A}_{\mathrm{ideal}}$ that receives only $\widetilde{h}$ and has no access to backdoored examples. In both (1) and (2), we let $f \sim \mathsf{P}$ and $\widetilde{h} = \mathrm{sgn}(\widetilde{f})$. In (1), we define the probability of generating a new backdoored example as:

$$p_{\mathrm{ideal}} = \mathbf{Pr}\left[\widetilde{f} \sim \mathsf{Backdoor}(f);\ (x, x') \sim \mathcal{A}_{\mathrm{ideal}}(\widetilde{h});\ \|x - x'\|_\infty \le \gamma, \widetilde{h}(x) \neq \widetilde{h}(x')\right]. \quad (1)$$

In the "real" world, the attacker has access to the model $\widetilde{h}$ as well as oracle access to $\mathsf{Activate}(\cdot, \mathsf{bk})$ to which the attacker can make polynomially many (potentially adaptively chosen) queries $x_1, \dots, x_q$, and receive the backdoored examples $\tilde{x}_i \leftarrow \mathsf{Activate}(x_i, \mathsf{bk})$ for each $i \in [q]$. In (2), we define the probability of generating a new backdoored example as:

$$p_{\mathrm{real}} = \mathbf{Pr}\left[(\widetilde{f}, \mathsf{bk}) \sim \mathsf{Backdoor}(f);\ (x, x') \sim \mathcal{A}_{\mathrm{real}}^{\mathsf{Activate}(\cdot,\mathsf{bk})}(\widetilde{h});\ \|x - x'\|_\infty \le \gamma, \widetilde{h}(x) \neq \widetilde{h}(x')\right]. \quad (2)$$

We mention that the notation $\mathcal{A}_{\mathrm{real}}^{\mathsf{Activate}(\cdot,\mathsf{bk})}$ means that the algorithm $\mathcal{A}_{\mathrm{real}}$ has oracle access to $\mathsf{Activate}(\cdot, \mathsf{bk})$. We define non-replicability as:

**Definition 17** (Non-Replicable Backdoor Goldwasser et al. [2022])**.** *For any security parameter $\lambda > 0$, we say that a backdoor* ($\mathsf{Backdoor}, \mathsf{Activate}$) *is non-replicable if for every polynomial function $q = q(\lambda)$ and every probabilistic polynomial-time $q$-query* admissible[4] *adversary $\mathcal{A}_{\mathrm{real}}$, there is a probabilistic polynomial-time adversary $\mathcal{A}_{\mathrm{ideal}}$ such that the following holds: $p_{\mathrm{real}} - p_{\mathrm{ideal}} \le \mathsf{negl}(\lambda)$, where the probabilities are defined in* (1) *and* (2).

### A.5 Boolean Circuits

In Section 4, we will need the following standard definition.

**Definition 18** ((Synchronous) Boolean Circuit)**.** *A Boolean circuit for $C : \{0,1\}^n \to \{0,1\}$ is a directed acyclic graph (DAG) where nodes represent Boolean operations (AND, OR, NOT) and edges denote operational dependencies that computes $C$, where $n$ is the number of input nodes.*

*A Boolean circuit is* synchronous *if all gates are arranged into layers, and inputs must be at the layer 0, i.e., for any gate $g$, all paths from the inputs to $g$ have the same length.*

## B Details of Tool #2: From Boolean Circuits to Neural Networks and Back

We now introduce two standard transformations: we define the transformation $T_k$ that discretizes a continuous bounded vector using $k$ bits of precision and $T^{-1}$ that takes a binary string and outputs a real number.

**Definition 19** (Real $\rightleftarrows$ Binary Transformation)**.** *Let $x \in [0,1]^n$, and let $k$ be a precision parameter. Define the transformation $T_k : [0,1]^n \to \{0,1\}^{n \cdot k}$ by the following procedure: For each component $x_i$ of $x$, represent $x_i$ as a binary fraction and extract the first $k$ bits after the binary point and denote this binary vector by $b_i \in \{0,1\}^k$, $i \in [n]$. Then $T_k(x)$ outputs $b = (b_1, \dots, b_n) \in \{0,1\}^{n \cdot k}$. Also, given a binary vector $b = (b_1, \dots, b_m) \in \{0,1\}^m$, define the inverse transformation $T^{-1} : \{0,1\}^m \to [0,1]$ by $T^{-1}(b) = \sum_{i=1}^m b_i/2^i$.*

We will also need the standard notion of size of a model.

**Definition 20** (Size of ANN & Boolean Circuits)**.** *Given an ANN $f$, we denote by $\mathrm{sz}(f)$ the size of $f$ and define it to be the bit complexity of each parameter. The size of a Boolean circuit $C$, denote by $\mathrm{sz}(C)$ is simply the number of gates it has.*

For example, an ANN that stores its parameters in 64 bits and has $M$ parameters has size $64 \cdot M$. We now present our first transformation which given $f : [0,1]^n \to [0,1]$ finds a Boolean circuit of small size that well-approximates $f$ in the following sense:

**Theorem 21** (ANN to Boolean)**.** *Given an $L$-Lipshitz ANN $f : [0,1]^n \to [0,1]$ of size $s$, then for any precision parameter $k \in \mathbb{N}$, there is an algorithm that runs in time $\mathrm{poly}(s, n, k)$ and outputs a*

[4] $\mathcal{A}_{\mathrm{real}}$ is *admissible* if $x' \notin \{x'_1, \dots, x'_q\}$ where $x'_i$ are the outputs of $\mathsf{Activate}(\cdot; \mathsf{bk})$ on $\mathcal{A}_{\mathrm{real}}$'s queries.

*Boolean circuit* $C : \{0,1\}^{n \cdot k} \to \{0,1\}^m$ *with number of gates* $\mathrm{poly}(s,n,k)$ *and* $m = \mathrm{poly}(s,n,k)$ *such that for any* $x, x'$*:*

$$|f(x) - T^{-1}(C(T_k(x)))| \le \frac{L}{2^k},$$

$$|T^{-1}(C(T_k(x))) - T^{-1}(C(T_k(x')))| \le \frac{L}{2^{k-1}} + L \cdot \|x - x'\|_\infty,$$

*where* $T_k$ *and* $T^{-1}$ *are defined in Definition 19.*

Let us provide some intuition regarding $T^{-1} \circ C \circ T_k$. Given $x \in [0,1]^n$, the transformation $T^{-1}(C(T_k(x)))$ involves three concise steps:

1. **Truncation** ($T_k$): Converts real input $x$ to its binary representation, keeping only the $k$ most significant bits.
2. **Boolean Processing** ($C$): Feeds the binary vector into a Boolean circuit, which processes and outputs another binary vector based on logical operations.
3. **Conversion to Real** ($T^{-1}$): Transforms the output binary vector back into a real number by interpreting it as a binary fraction.

For the proof of Theorem 21, see Appendix H.1. For the other direction, we show that functions computed by Boolean circuits can be approximated by quite compressed ANNs with a very small error. Function approximation by neural networks has been studied extensively (see Appendix C for a quick overview). Our approach builds on [Fearnley et al., 2022, Section E]. The proof appears in Appendix H.2.

**Theorem 22** (Boolean to ANN, inspired by Fearnley et al. [2022])**.** *Given a Boolean circuit* $C : \{0,1\}^{n \cdot k} \to \{0,1\}^m$ *with* $k, m, n \in \mathbb{N}$ *with* $M$ *gates and* $\epsilon > 0$ *such that*

$$|T^{-1}(C(T_k((x))) - T^{-1}(C(T_k(x')))| \le \epsilon \qquad \forall x, x' \in [0,1]^n \text{ s.t. } \|x - x'\|_\infty \le \frac{1}{2^k},$$

*where* $T_k$ *and* $T^{-1}$ *are defined in Definition 19, there is an algorithm that runs in time* $\mathrm{poly}(n,k,M)$ *and outputs an ANN* $f : [0,1]^n \to [0,1]$ *with size* $\mathrm{poly}(n,k,M)$ *such that for any* $x \in [0,1]^n$ *it holds that* $|T^{-1}(C(T_k(x))) - f(x)| \le 2\epsilon$.

## C Related Work

**Comparison with Goldwasser et al. [2022]** The work of Goldwasser et al. [2022] is the closest to our work. At a high level, they provide two sets of results. Their first result is a black-box undetectable backdoor. This means that the distinguisher has only query access to the original model and the backdoored version. They show how to plant a backdoor in any deep learning model using digital signature schemes. Their construction guarantees that, given only query access, it is computationally infeasible, under standard cryptographic assumptions, to find even a single input where the original model and the backdoored one differ. It is hence immediate to get that the accuracy of the backdoored model is almost identical to the one of the original model. Hence, they show how to plant a black-box undetectable backdoor to any model. Their backdoor is also non-replicable. Our result applies to the more general scenario of white-box undetectability and hence is not comparable.

The second set of results in Goldwasser et al. [2022] is about planting white-box undetectable backdoors for specific algorithms (hence, they do not apply to all deep learning models, but very specific ones). The main model that their white-box attacks apply to is the RFF model of Rahimi and Recht [2007]. Let us examine how Goldwasser et al. [2022] add backdoors that are white-box undetectable. They first commit to a parameterized model (in particular, the Random Fourier Features (RFF) model of Rahimi and Recht [2007] or a random 1-layer ReLU NN), and then the honest algorithm commits to a random initialization procedure (e.g., every weight is sampled from $\mathcal{N}(0, I)$). After that, the backdoor algorithm samples the initialization of the model from an "adversarial" distribution that is industinguishable from the committed honest distribution and then uses the committed train procedure (e.g., executes the RFF algorithm faithfully on the given training data). Their main result is that, essentially, they can plant a backdoor in RFF that is white-box undetectable under the hardness of the Continuous Learning with Errors (CLWE) problem of Bruna et al. [2021]. Our result aims to achieve further generality: we show that *any* training procedure followed by an obfuscation step can be backdoored in a white-box and non-replicable manner.

**Other related works** The work of Moitra et al. [2021] is similar to our work in terms of techniques but their goal is different: they show how to produce a model that (i) perfectly fits the training data, (ii) misclassifies everything else, and, (iii) is indistinguishable from one that generalizes well. At a technical level, Moitra et al. [2021] also use indistinguishability obfuscation and signature schemes. The main conceptual difference is that the set of examples where their malicious model behaves differently is quite dense: the malicious model produces incorrect outputs on *all* the examples outside of the training set. In our setting and that of Goldwasser et al. [2022], the changes in the model's behavior are essentially measure zero on the population level and a backdoored model generalizes exactly as the original model.

Hong et al. [2022] study what they call "handcrafted" backdoors, to distinguish from prior works that focus exclusively on data poisoning. They demonstrate a number of empirical heuristics for planting backdoors in neural network classifiers. Garg et al. [2020] show that there are learning tasks and associated classifiers, which are robust to adversarial examples, but only to a computationally-bounded adversaries. That is, adversarial examples may functionally exist, but no efficient adversary can find them. Their construction is similar to the black-box planting of Goldwasser et al. [2022]. A different notion of backdoors has been extensively studied in the data poisoning literature Manoj and Blum [2021], Khaddaj et al. [2023], Jha et al. [2024], Hayase et al. [2021], Tran et al. [2018], Chen et al. [2017], Gu et al. [2019]. In this case, one wants to modify some part of the training data (and their labels) to plant a backdoor in the final classifier, without tampering with any other part of the training process. See also Salman et al. [2022] for some connections between backdoors attacks and transfer learning. On the other side, there are various works studying backdoor detection Alex et al. [2023].

The line of work on adversarial examples Ilyas et al. [2019], Athalye et al. [2018], Szegedy et al. [2013] is also relevant to backdoors. Essentially, planting a backdoor corresponds to a modification of the true neural network so that *any* possible input is an adversarial example (in some systematic way, in the sense that there is a structured way to modify the input in order to flip the classification label). Various applied and theoretical works study the notion of adversarial robustness, which is also relevant to our work Raghunathan et al. [2018], Wong and Kolter [2018], Shafahi et al. [2019], Bubeck et al. [2019]. Finally, backdoors have been extensively studied in cryptography. Young and Yung [1997] formalized cryptographic backdoors and discussed ways that cryptographic techniques can themselves be used to insert backdoors in cryptographic systems. This approach is very similar to both Goldwasser et al. [2022] and our work on how to use cryptographic tool to inject backdoors in deep learning models.

**Approximation by Neural Networks** There is a long line of research related to approximating functions by ANNs. It is well-known that sufficiently large depth-2 neural networks with reasonable activation functions can approximate any continuous function on a bounded domain Cybenko [1989], Barron [1993, 1994]. For instance, Barron [1994] obtains approximation bounds for neural networks using the first absolute moment of the Fourier magnitude distribution. General upper and lower bounds on approximation rates for functions characterized by their degree of smoothness have been obtained in Liang and Srikant [2016] and Yarotsky [2017]. Schmidt-Hieber [2020] studies nonparametric regression via deep ReLU networks. Hanin and Sellke [2017] establish universality for deep and fixed-width networks. Depth separations have been exhibited e.g., by Eldan and Shamir [2016], Safran and Shamir [2017], Telgarsky [2016]. Lu et al. [2017], Savarese et al. [2019] study how width affects the expressiveness of neural networks. For further related work, we refer to DeVore et al. [2021], Daubechies et al. [2022], Telgarsky [2021]. In our result (cf. Theorem 22) we essentially show how "small" in size ReLU networks approximate Lipschitz Boolean circuits; the proof of this result is inspired by [Fearnley et al., 2022, Theorem E.2]. We note that our result could be extended so that any polynomially-approximately-computable class of functions (as in Fearnley et al. [2022]) can be approximated by "small" in size ReLU networks. Abbe and Sandon [2020] considers the case of binary classification in the Boolean domain and shows how to convert any poly-time learner in a function learned by a poly-size neural net trained with SGD on a poly-time initialization with poly-steps, poly-rate and possibly poly-noise.

**Backdoors in LMs, Watermarking and Steganography** Vulnerabilities of language models in backdoor attacks have been raised as an important - yet under-explored - problem in Anwar et al. [2024]. In our work, we make theoretical progress on this question. Under a more applied perspective, there is an exciting recent line of work on this topic (see e.g., Xu et al. [2022], Kandpal et al. [2023], Xiang et al. [2024], Wang et al. [2023], Zhao et al. [2023, 2024], Rando and Tramèr [2023], Rando et al. [2024], Hubinger et al. [2024], Li et al. [2024], Huang et al. [2023], Yang et al. [2024], Shen et al. [2021], Wen et al. [2024], Cai et al. [2022], Mei et al. [2023], Xu et al. [2023], Wan et al. [2023], He et al. [2024] and the references therein). Our approach relies on steganography, the method of concealing a message within another message, see e.g., Anderson and Petitcolas [1998], Hopper et al. [2002], de Witt et al. [2022], Dedić et al. [2005], Kaptchuk et al. [2021]. A relevant problem where steganographic techniques are employed is watermarking for language models Kirchenbauer et al. [2023]. Watermarking in LLMs Aaronson [2023] is extensively studied recently. We now mention relevant theoretical works. Christ et al. [2023] provide watermarks for language models which are computationally undetectable, in the following sense: the watermarks can be detected only with the knowledge of a secret key; without it, it is computationally intractable to distinguish watermarked outputs from the original ones. Note that this notion of undetectability is exactly the same as our Definition 14 of "computational indistinguishability". Zamir [2024] uses steganography to hide an arbitrary secret payload in the response of an LLM. This approach is closely related to our work but has a different objective. Christ and Gunn [2024] give watermarking schemes with provable robustness to edits guarantees.

## D Obfuscation in the Honest Pipeline

Obfuscation is a technique commonly employed in software applications to enhance the robustness of models against malicious attacks. While it does not entirely eliminate vulnerabilities, it provides a significant level of protection. In principle, as articulated by Barak et al. [2001],

*"roughly speaking, the goal of (program) obfuscation is to make a program 'unintelligible' while preserving its functionality. Ideally, an obfuscated program should be a 'virtual black box,' in the sense that anything one can compute from it one could also compute from the input-output behavior of the program."*

Hence, obfuscation serves to downgrade the power of an adversarial entity from having white-box access to a model, which entails full transparency, to a scenario where the adversary has roughly speaking black-box access, i.e., where the internal workings of the model remain concealed.

Intellectual Property (IP) and Privacy attacks represent critical categories of malicious threats, against which the application of obfuscation is expected to enhance the system's resilience.

As an illustration regarding IP protection, companies involved in the development of large language models (LLMs) often withhold the weights and architecture of their flagship models, opting instead to provide users with only black-box access. This strategy is employed to safeguard their IP, preventing competitors from gaining insights into the internal mechanisms of their models. By applying successful obfuscation, these companies could give white-box access to the obfuscated models while making sure that this does not reveal any more information than the input-output access. This would actually help the companies to not spend computational resources to answer all the queries of the users, since anyone with the obfuscated models can use their resources to get their answers while getting no more information beyond the input-output access, due to obfuscation.

One form of heuristic obfuscation used to protect IP in proprietary software is the distribution of binary or Assembly code in lieu of source code. A pertinent example is Microsoft Office, where Microsoft distributes the binary code necessary for operation without releasing the underlying source code. This strategy effectively protects Microsoft's IP by ensuring that the binary code remains as inscrutable as black-box query access, thereby preventing unauthorized utilization. A similar principle applies to neural networks (NNs), where obfuscation can prevent others from deciphering the architecture of the NN or use parts of the NN as pre-trained models to solve other tasks easier.

Turning to privacy attacks, it is evident that black-box access to a model is substantially more restrictive than white-box access. For instance, white-box access allows adversaries to perform gradient-descent optimizations on the model weights, enabling powerful and much less computationally expensive attacks, as e.g., demonstrated in Carlini and Farid [2020].

However, it is important to note that obfuscation does not inherently defend against privacy attacks that exploit the input-output behavior of a model rather than its internal structure, such as model inversion and membership inference attacks. These privacy concerns require specialized defenses, such as differential privacy Rahman et al. [2018]. Nonetheless, these defenses are rendered ineffective if an adversary gains white-box access to the model Zhang et al. [2020], Carlini et al. [2023], Nasr et al. [2019].

## E Backdoor Planting in Language Models

Vulnerability of language models to backdoors is a challenging problem, raised e.g., in Anwar et al. [2024] and studied experimentally in various works Kandpal et al. [2023], Xiang et al. [2024], Wang et al. [2023], Zhao et al. [2023, 2024], Rando and Tramèr [2023], Hubinger et al. [2024]. We initiate a theoretical study of planting backdoors to language models (LMs); we now discuss how to apply our techniques of Section 5 to language models. We first introduce the notion of planting a backdoor in a language model (Definition 27): we assume a dual model configuration consisting of an honest model $f$ and a malicious model $\widehat{f}$, with a trigger activation mechanism (see Section 5.1 for details). This mechanism allows for covert signals to be embedded within the model's outputs, activating the backdoor under specific conditions without altering the apparent meaning of the text. The main difference between this approach and the attack in ANNs (Section 5) is the implementation of the trigger mechanism. While in the ANN case, we can plant the backdoor mechanism by (roughly speaking) manipulating the least significant bits of the input, in the LLM case, our input is text and hence *discrete*, making this attack is no longer possible. Our conceptual idea is that if we assume access to a *steganographic function* Shih [2017], we can implement a trigger mechanism. We refer to Appendix E.2 for details. Using this approach combined with our tools of Section 4 we obtain the attack presented in Appendix E.2.2. We now continue with some background on LMs.

### E.1 Background on Language Models

We start this background section by defining the crucial notion of a *token*. In natural language processing, a token is the basic unit of text processed by models. Tokens are generated from raw text through a procedure called tokenization, which breaks down extensive textual data into manageable parts. These tokens vary in granularity from characters to subwords and complete words, depending on the tokenization method employed. The entire set of tokens that a model can utilize is called the vocabulary and is denoted by $\mathcal{T}$ (see Definition 23).

**Definition 23** (Token and Tokenization)**.** *A* token *is the atomic element of text used in natural language processing and is denoted as an element in a finite set $\mathcal{T}$.* Tokenization *is the process of decomposing a string of characters from an alphabet $\Sigma$ into a sequence of tokens, defined by a function $\tau : \Sigma^* \to \mathcal{T}^*$.*

Autoregressive language models leverage sequences of tokens to generate text. These models are typically implemented as ANNs that approximate the conditional probability distribution of the next token based on the preceding sequence. We provide the following formal definition, under the assumption that the token window of the model is bounded and equal to $k$.

**Definition 24** ((Autoregressive) Language Model)**.** *For a number $k \in \mathbb{N}$, a* language model (LM) *is a function $f : \mathcal{T}^k \to \Delta(\mathcal{T})$ that maps a sequence of $k$ tokens $\boldsymbol{t}_0$ (with potentially padded empty tokens) to a distribution over the output tokens; given an initial sequence of tokens $\boldsymbol{t}_0 \in \mathcal{T}^k$ as input, an autoregressive language model uses $f$ to generate each token $t_k$ in an auto-regressive manner e.g., the conditional probability that the $m$-th generated token is $t_m$ is:*

$$P(t_m | \boldsymbol{t}_0 \leftarrow t_1, t_2, \ldots, t_{m-1}) = f(\boldsymbol{t}_0, t_1, t_2, \ldots, t_{m-1}),$$

*where we denote by $(\boldsymbol{t}_0 \leftarrow t_1, t_2, \ldots, t_{m-1}) \in \mathcal{T}^k$ the token of length $k$ where we replace empty padded tokens in $\boldsymbol{t}_0$ with token sequence $t_1, t_2, \ldots, t_{m-1}$.*[5] *This model predicts $t_m$ by sampling from the distribution iteratively, constructing a text sequence one token at a time.*

### E.2 Trigger Activation Mechanism: Steganography in Language Models

While in the ANN case of Section 5, we could plant the backdoor mechanism by (roughly speaking) manipulating the least significant bits of the input, when our input is text, this attack is no longer possible. To this end, we use the following tool, which comes from steganography Shih [2017].

[5]If the length of the sequence exceeds $k$, we only use the last $k$ tokens of it.

**Definition 25** (Steganographic Function for Language Model)**.** *A* steganographic function *is a pair of functions $\sigma : \mathcal{T}^k \times \{0,1\}^M \to \mathcal{T}^k$ and $\sigma^{-1} : \mathcal{T}^k \to \{0,1\}^M$, for some $M \in \mathbb{N}$. The function $\sigma$ takes a prompt $x \in \mathcal{T}^k$ and a secret binary code $m \in \{0,1\}^M$, and outputs an altered prompt $x' \in \mathcal{T}^k$. The inverse function $\sigma^{-1}$ satisfies the property that it outputs the secret code $m$ given the altered prompt $x'$:*

$$\sigma^{-1}(\sigma(x,m)) = m \quad \forall x \in \mathcal{T}^k, m \in \{0,1\}^M. \tag{3}$$

Steganography is the practice of concealing a message within another message, see e.g., Hopper et al. [2002], de Witt et al. [2022], Dedić et al. [2005]. In the context of language models, steganography can be used to embed covert signals within the outputs (e.g., text) generated by the model. Steganography was previously used for watermarking purposes, see e.g., Christ et al. [2023], Zamir [2024], Christ and Gunn [2024] and the references therein. These signals can serve as activation mechanisms without altering the apparent meaning of the text. Implementing a steganographic function is beyond the scope of our paper; we assume its existence and functionality, providing only an illustrative example as a possible approach in Example 26.

**Example 26.** *Consider implementing a basic dictionary where each token (word or subword) has associated typos that can be deliberately introduced or avoided. Each typo variation corresponds to a binary code—'0' might represent the standard spelling, while '1' could represent a version with a common typo. By controlling the presence of these typos in the output text, the model can embed a binary message within a seemingly normal text. This method allows for the transmission of hidden information that is undetectable to casual observers but can be decoded by those aware of the steganographic scheme.*

### E.2.1 Defining Backdoored Language Models

In this section, we formally introduce the concept of backdooring a language model. A backdoor in a language model allows for a hidden behavior to be triggered under specific conditions, while the model otherwise behaves normally. We assume the existence of a steganographic function $\sigma : \mathcal{T}^k \times \{0,1\}^M \to \mathcal{T}^k$ that encodes a hidden activation trigger into an input prompt $\boldsymbol{t} \in \mathcal{T}^k$, producing a modified prompt $\boldsymbol{t}' \in \mathcal{T}^k$. The backdoored language model $f$ utilizes this function to introduce hidden behaviors triggered by specific conditions on the hidden message $\boldsymbol{t}$. The definition is formally stated below.

**Definition 27** (Backdoored Language Model)**.** *A language model $\widetilde{f}$ that originally intends to implement language model $f$ is said to be backdoored if there exists another language model $\widehat{f}$, a steganographic function $\sigma$ and its inverse $\sigma^{-1}$, and a condition $E$ such that:*

1. *$\widetilde{f}$ behaves as $f$ under normal circumstances:*

$$\forall \boldsymbol{t} \in \mathcal{T}^k : E(\sigma^{-1}(\boldsymbol{t})) = 0 \implies \widetilde{f}(\boldsymbol{t}) \approx f(\boldsymbol{t}). \tag{4}$$

2. *$\widehat{f}$ exhibits altered behavior if the condition $E$ on the prompt $\boldsymbol{t}$ is satisfied:*

$$\forall \boldsymbol{t} \in \mathcal{T}^k : E(\sigma^{-1}(\boldsymbol{t})) = 1 \implies \widetilde{f}(\boldsymbol{t}) \approx \widehat{f}(\boldsymbol{t}). \tag{5}$$

It is important to note that it is not immediately clear what the right generalization of a non-replicable backdoor should be for LLMs. In order to achieve non-replicability (e.g., in the classification setting) we partition the input into parts and designate a specific part for non-replicability via digital signatures. However, when the input (in our case, the tokens) is discrete, performing this sort of partition is complex because it is not clear e.g., which are the 'most significant' bits as in the classification case. Exploring non-replicability in the context of language models is left as future work.

### E.2.2 Embedding Undetectable Backdoors to a Language Model

Consider an insidious firm that wants to train a model such that it generates text from an insidious language model $\widehat{f}$ on triggered inputs. Unlike planting the backdoor in ANN (Section 5), since the input and output space of the language model is discrete, a language model $f$ can be easily implemented as a Boolean circuit (see Remark 28).

**Remark 28.** *Consider a language model operating on a token set $\mathcal{T}$. Without loss of generality, we can represent each token in the discrete set $\mathcal{T}$ by its one-hot embedding in a vector of dimension $|\mathcal{T}|$. Given this representation, the operations performed by a language model can be directly mapped to a Boolean circuit with size $O(\mathrm{sz}(f), |\mathcal{T}|)$. Conversely, the outputs of a Boolean circuit $C$, can be interpreted as one-hot vectors, thus allowing us to transform it to a ANN $f$ with size $O(\mathrm{sz}(C))$.*

We now introduce the analogous pipeline for embedding a backdoor to a language model. Our construction is tuned by security parameters $\lambda, \lambda_1 > 0$:

1. **Training the LM.** Begin by training a language model $f$ in a standard fashion so that it accurately models the function $f : \mathcal{T}^k \to \Delta(\mathcal{T}^k)$, representing the LLM's intended behavior. This corresponds to the honest training process.
2. **LM to Boolean Circuit Transformation.** Transform the language model $f$ into a Boolean circuit, denoted as $C : \{0,1\}^n \to \{0,1\}^m$ (see Remark 28).
3. **Part of the Insidious Pipeline.** Consider a malicious language model $\widehat{f}$. Let $\widehat{C}$ be the Boolean circuit corresponding to the model $\widehat{f}$.
   3.1. **Input Space Partitioning.** On prompt $\boldsymbol{t} \in \mathcal{T}^k$, use the original prompt $\boldsymbol{t}$ as $x_{\mathsf{C}}$ and the output of the inverse steganographic function $\sigma^{-1}(\boldsymbol{t})$ as $x_{\mathsf{BD}}$.
   3.2. **Backdoor Implementation.** Use Section 4.1 with security parameters $\lambda_1$ and $\lambda_2 = 0$ to inject a backdoor within $x_{\mathsf{BD}}$, with sampled seed $s^* \in \{0,1\}^{\lambda_1}$. Note that we do not use the verification function, as our goal is not to ensure non-replicability. Denote the resulting backdoored Boolean circuit by $C_{int}$.
4. **Application of Indistinguishability Obfuscation (iO).** Apply indistinguishability obfuscation with security parameter $\lambda$ to $C_{int}$, and let $\widetilde{C}$ be the outputted Boolean circuit (Definition 7).
5. **Circuit to LM Transformation.** Convert the obfuscated Boolean circuit $\widetilde{C}$ back to a LM $\widetilde{f} : \mathcal{T}^k \to \Delta(\mathcal{T}^k)$ (see Remark 28).

The above procedure defines the behavior of the honest and the malicious algorithm on input a language model $f$:

- The honest obfuscation pipeline corresponds to the sequential composition of steps $(2), (4)$ and $(5)$.
- The backdoor attack corresponds to the sequential composition of steps $(2), (3), (4)$ and $(5)$.

To activate the backdoor on original prompt $\boldsymbol{t}$, we use the steganographic function $\sigma$ on the intended seed $s^*$, e.g., the modified prompt $\boldsymbol{t}'$ is $\sigma(\boldsymbol{t}, s^*)$. Note that we set $x_{\mathsf{BD}}$ in the above procedure to be equal to the secret seed $s^*$.

**Theorem 29** (Planting Backdoor to Obfuscated LM)**.** *Let $n, \mathcal{T}$ as introduced in Steps (1)-(2) above. Assume that one-way functions and computationally-secure and efficient indistinguishability obfuscators for polynomial-sized circuits (as in Assumption 8) both exist. Further assume that a steganographic function of Definition 25 exists. Given security parameters $\lambda, \lambda_1 \in \mathbb{N}$ and a trained language model $f$ with honest obfuscation $\widetilde{f}$ and with a malicious language model $\widehat{f}$, there exists a backdoor attack* $(\mathsf{Backdoor}, \mathsf{Activate})$ *described in Steps (1)-(5) above so that the following are true:*

1. *The backdoor runs in time $\mathrm{poly}(n, \mathrm{sz}(C), \mathrm{sz}(\widehat{C}), |\mathcal{T}|, \lambda, \lambda_1)$, where $C$ (resp. $\widehat{C}$) are the Boolean circuits induced by $f$ (resp. $\widehat{f}$).*
2. *$\widetilde{f}$ and $f' \sim \mathsf{Backdoor}$ are white-box undetectable.*
3. *For any input $\mathbf{t} \in \mathcal{T}^k$ transformed into $\mathbf{t}' \in \mathcal{T}^k$ to activate the backdoor $f' \sim \mathsf{Backdoor}$, it satisfies:*

$$f'(\boldsymbol{t}') = \widehat{f}(\boldsymbol{t}').$$

The last Item holds since $\boldsymbol{t}'$ corresponds to $\sigma(\boldsymbol{t}, s^*)$ and so $\sigma^{-1}(\boldsymbol{t}') = s^*$, which will appear in $x_{\mathsf{BD}}$, thus activating the backdoor. The proof is quite short and appears in Appendix I.2.

## F Conclusion & Open Questions

Given the plethora of applications of Machine Learning in general, and neural networks in particular, questions regarding the trustworthiness of publicly released models naturally arise. In particular, before deploying a neural network we need to guarantee that no backdoors have been injected allowing bad actors to arbitrarily control the model behavior. In this paper, we investigate the existence of backdoor attacks to obfuscated neural networks which are undetectable even when given white-box access. The notion of obfuscation that we consider is the well-studied and mathematically founded indistinguishability obfuscation (iO). We also show how our techniques can inspire backdoor schemes in large language models when combined with ideas from steganography.

While our constructions are purely theoretical, we leave as an interesting direction how to use heuristic obfuscation methods to show practical instantiations of our constructions. Another interesting open question is whether cryptographic schemes weaker than iO suffice to show backdoor undetectability in the white-box model.

## G Proof of Theorem 10

We restate the Theorem for convenience. [White-Box Undetectability via iO] Assuming the existence of secure pseudorandom generators (Assumption 6) and secure indistinguishability obfuscation (Assumption 8), for any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$, and security parameters $\lambda, \lambda_1, \lambda_2 \in \mathbb{N}$ it holds that

$$\left|\mathbf{Pr}\left[\mathcal{A}(i\mathcal{O}(1^\lambda, C)) = 1\right] - \mathop{\mathbf{Pr}}_{s^* \sim U\{0,1\}^{\lambda_1}}\left[\mathcal{A}(i\mathcal{O}(1^\lambda, \widetilde{C})) = 1\right]\right| \leq \mathsf{negl}(\lambda_3) + \mathsf{negl}(\lambda_1).$$

*Proof.* We consider a family of circuits $C_r$, parameterized by $r \in \{0,1\}^{2\lambda_1}$ that implement the following function:

$$\begin{aligned} f_r(x_{\mathsf{C}}, (x_{\mathsf{PRG}}, x_V)) =& \widehat{C}(\boldsymbol{x}_C) \cdot \mathbb{1}[\mathsf{PRG}(x_{\mathsf{PRG}}) = \mathsf{PRG}(r)] \cdot \mathsf{Verify}(pk, x_{\mathsf{C}}, x_V) \\ &+ C(\boldsymbol{x}) \cdot (1 - \mathbb{1}[\mathsf{PRG}(x_{\mathsf{PRG}}) = \mathsf{PRG}(r)] \cdot \mathsf{Verify}(pk, x_{\mathsf{C}}, x_V)). \end{aligned}$$

By security of PRG (Assumption 6), for any PPT $\mathcal{A}$:

$$\left|\mathop{\mathbf{Pr}}_{s^* \sim U\{0,1\}^{\lambda_1}}\left[\mathcal{A}(\mathsf{PRG}(s^*)) = 1\right] - \mathop{\mathbf{Pr}}_{r^* \sim U\{0,1\}^{2\lambda_1}}\left[\mathcal{A}(r^*) = 1\right]\right| \leq \mathsf{negl}(\lambda_1).$$

By further restricting to PPT algorithms $\mathcal{A}$ that operate on the composed input $T(r) = i\mathcal{O}(1^\lambda, C_r)$ , and since both compositions take poly($|C|$), we have for any PPT algorithm $\mathcal{A}'$:

$$\left|\mathop{\mathbf{Pr}}_{s^* \sim U\{0,1\}^{\lambda}}\left[\mathcal{A}'(i\mathcal{O}(1^\lambda, C_{\mathsf{PRG}(s^*)})) = 1\right] - \mathop{\mathbf{Pr}}_{r^* \sim U\{0,1\}^{2\lambda}}\left[\mathcal{A}'(i\mathcal{O}(1^\lambda, C_{r^*})) = 1\right]\right| \leq \mathsf{negl}(\lambda_1). \quad (6)$$

Since $|\text{range}(\mathsf{PRG})| \leq 2^{\lambda_1} = \mathsf{negl}(\lambda_1)$, and

$$\begin{aligned} f_{r^*}(x_{\mathsf{C}}, (x_{\mathsf{PRG}}, x_V)) =& C(\boldsymbol{x}) \cdot (1 - \mathbb{1}[\mathsf{PRG}(x_{\mathsf{PRG}}) = \mathsf{PRG}(r^*)] \cdot \mathsf{Verify}(pk, x_{\mathsf{C}}, x_V)) \\ &+ \widehat{C}(\boldsymbol{x}_C) \cdot \mathbb{1}[\mathsf{PRG}(x_{\mathsf{PRG}}) = \mathsf{PRG}(r^*)] \cdot \mathsf{Verify}(pk, x_{\mathsf{C}}, x_V), \end{aligned}$$

$$\begin{aligned} &\mathop{\mathbf{Pr}}_{r^* \sim U\{0,1\}^{2\lambda_1}}[\nexists s \in \{0,1\}^{\lambda_1} : \mathsf{PRG}(s) = r] \geq 1 - \mathsf{negl}(\lambda_1) \\ \Rightarrow &\mathop{\mathbf{Pr}}_{r^* \sim U\{0,1\}^{2\lambda_1}}[C_{r^*}(x) = C(x) \forall x \in \{0,1\}^n] \geq 1 - \mathsf{negl}(\lambda_1). \end{aligned}$$

Hence with probability at least $1 - 2^{\lambda_1}$, circuits $C_{r^*}$ and $C$ are computationally equivalent and hence by application of iO we further have:

$$\left|\mathbf{Pr}\left[\mathcal{A}(i\mathcal{O}(1^\lambda, C)) = 1\right] - \mathop{\mathbf{Pr}}_{r^* \sim U\{0,1\}^{\lambda_1}}\left[\mathcal{A}(i\mathcal{O}(1^\lambda, C_{r^*})) = 1\right]\right| \leq \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda_1). \quad (7)$$

We conclude by noticing that circuit $C_{\mathsf{PRG}(s^*)}$ is identically equal to circuit $\widetilde{C}(x)$, and combining (6) and (7):

$$\left| \underset{s^* \sim U\{0,1\}^{\lambda_1}}{\mathbf{Pr}} \left[\mathcal{A}'(i\mathcal{O}(1^\lambda, C_{\mathsf{PRG}(s^*)})) = 1\right] - \underset{r^* \sim U\{0,1\}^{2\lambda}}{\mathbf{Pr}} \left[\mathcal{A}'(i\mathcal{O}(1^\lambda, C_{r^*})) = 1\right] \right| \leq \mathsf{negl}(\lambda_1)$$

$$\Rightarrow \left| \mathbf{Pr}\left[\mathcal{A}(i\mathcal{O}(1^\lambda, C)) = 1\right] - \underset{s^* \sim U\{0,1\}^{\lambda_1}}{\mathbf{Pr}} \left[\mathcal{A}(i\mathcal{O}(1^\lambda, \widetilde{C})) = 1\right] \right| \leq \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda_1).$$

□

## H Proofs of Section 4.2

### H.1 Proof of Theorem 21

Let us restate the result. [ANN to Boolean] Given an $L$-Lipshitz ANN $f : [0,1]^n \to [0,1]$ of size $s$, then for any precision parameter $k \in \mathbb{N}$, there is an algorithm that runs in time $\text{poly}(s, n, k)$ and outputs a Boolean circuit $C : \{0,1\}^{n \cdot k} \to \{0,1\}^m$ with number of gates $\text{poly}(s, n, k)$ and $m = \text{poly}(s, n, k)$ such that for any $x, x'$:

$$|f(x) - T^{-1}(C(T_k(x)))| \leq \frac{L}{2^k},$$

$$|T^{-1}(C(T_k(x))) - T^{-1}(C(T_k(x')))| \leq \frac{L}{2^{k-1}} + L \cdot \|x - x'\|_\infty,$$

where $T_k$ and $T^{-1}$ are defined in Definition 19.

*Proof.* The transformation of Theorem 21 follows by simply compiling a neural network to machine code (see also [Shi et al., 2020, Section 3]) where the input is truncated within some predefined precision. Note that

$$|f(x) - T^{-1}(C(T_k(x)))| \leq L \cdot \|x - T^{-1}(T_k(x))\|_\infty = \frac{L}{2^k}$$

and we also have $|T^{-1}(C(T_k(x))) - T^{-1}(C(T_k(x')))| \leq |T^{-1}(C(T_k(x))) - f(x)| + |T^{-1}(C(T_k(x'))) - f(x')| + |f(x) - f(x')| \leq \frac{L}{2^{k-1}} + L \cdot \|x - x'\|_\infty$. □

### H.2 Proof of Theorem 22

We first restate the Theorem we would like to prove. [Boolean to ANN, inspired by Fearnley et al. [2022]] Given a Boolean circuit $C : \{0,1\}^{n \cdot k} \to \{0,1\}^m$ with $k, m, n \in \mathbb{N}$ with $M$ gates and $\epsilon > 0$ such that

$$|T^{-1}(C(T_k((x))) - T^{-1}(C(T_k(x')))| \leq \epsilon \qquad \forall x, x' \in [0,1]^n \text{ s.t. } \|x - x'\|_\infty \leq \frac{1}{2^k},$$

where $T_k$ and $T^{-1}$ are defined in Definition 19, there is an algorithm that runs in time $\text{poly}(n, k, M)$ and outputs an ANN $f : [0,1]^n \to [0,1]$ with size $\text{poly}(n, k, M)$ such that for any $x \in [0,1]^n$ it holds that $|T^{-1}(C(T_k(x))) - f(x)| \leq 2\epsilon$.

*Proof.* Our proof is directly inspired by Fearnley et al. [2022]. We start with the definition of an arithmetic circuit (see Fearnley et al. [2022] for details). An *arithmetic circuit* representing the function $f : \mathbb{R}^n \to \mathbb{R}^m$ is a circuit with $n$ inputs and $m$ outputs where every internal node is a gate with fan-in 2 performing an operation in $\{+, -, \times, \max, \min, >\}$ or a rational constant (modelled as a gate with fan-in 0). *Linear arithmetic circuits* are only allowed to use the operations $\{+, -, \max, \min, \times\zeta\}$ and rational constants; the operation $\times\zeta$ denotes multiplication by a constant. Note that every linear arithmetic circuit is a well-behaved arithmetic circuit (see Fearnley et al. [2022]) and hence can be evaluated in polynomial time. Fearnley et al. [2022] show that functions computed by arithmetic circuits can be approximated by linear arithmetic circuits with quite small error. We will essentially show something similar replacing linear arithmetic circuits with ReLU networks.

Our proof proceeds in the following three steps, based on [Fearnley et al., 2022, Section E].

**Discretization** Let $N = 2^k$. We discretize the set $[0,1]$ into $N+1$ points $\mathcal{I} = \{0, 1/N, 2/N, \dots, 1\}$, and for any element $p \in [0,1]^n$, we let $\widehat{p} \in \mathcal{I}^n$ denote its discretization, i.e., $\widehat{p} = (\widehat{p}_i)_{i \in [n]}$ such that for each coordinate $i \in [n]$

$$\widehat{p}_i = \frac{i^*}{N}\,, \text{ where } i^* = \max\left\{ i^* \in [N] : \frac{i^*}{N} \le p_i \right\}. \tag{8}$$

**Construct Linear Arithmetic Circuit** Given as input a Boolean circuit $C$, our strategy is to use the approach of [Fearnley et al., 2022, Lemma E.3] to construct, in time polynomial in the size of $C$, a linear arithmetic circuit $F : [0,1]^n \to \mathbb{R}$ that will well-approximate $C$ as we will see below.

Before we proceed with the proof, we use the following gadget that approximates the transformation $T_k$.

**Theorem 30** (Bit Extraction Gadget Fearnley et al. [2022]). *Let* $\operatorname{proj}(x) = \min(0, \max(1, x))$ *and consider a precision parameter* $\ell \in \mathbb{N}$. *Define the bit extraction function*

$$\begin{aligned} t_0(x) &= 0\,, \\ t_k(x) &= \operatorname{proj}\left(2^\ell \cdot \left(x - 2^{-k} - \sum_{k'=0}^{k-1} 2^{-k'} \cdot t_{k'}(x)\right)\right), \qquad \textit{for } k > 0. \end{aligned}$$

*Fix* $k \in \mathbb{N}$. $t_k(x)$ *can be computed by a linear arithmetic circuit using* $O(k)$ *layers and a total of* $O(k^2)$ *nodes. The output is* $\hat{T}_k(x) = (t_0(x), \dots, t_k(x))$.

*Moreover, given a number* $x \in (0,1)$, *represented in binary as* $\{b_0.b_1b_2\dots\}$ *where* $b_{k'}$ *is the* $k'$*-th most significant digit, if there exists* $k^* \in [k+1, \ell]$ *such that* $b_{k^*} = 1$, *then* $\hat{T}_k(x) = \{0.b_1b_2\dots b_k\}$.

*Proof.* We prove the first part of the statement based on induction that for each $k$, we can compute a linear arithmetic circuit outputting $(x, f_0(x), f_1(x), \dots, f_k(x))$ using $3 \cdot k + 2$ layers a total of $3\sum_{k'=1}^{k} k' + 4$ nodes.

**Base case** $k = 0$**:** We can trivially design a linear arithmetic circuit using two layers and two nodes that outputs $(x, f_0(x)) = (x, 0)$.

**Induction step:** Assume that for $k' - 1$ we can design a linear arithmetic circuit that outputs

$$(x, f_0(x), f_1(x), \dots, f_{k'-1}(x))\,.$$

Let $C = 2^\ell \cdot \left(x - 2^{-k'} - \sum_{k'=0}^{k'-1} 2^{-k'} \cdot f_{k'}(x)\right)$. Observe that

$$f_{k'}(x) = \min(1, \max(C, 0))\,,$$

and thus we can extend the original linear arithmetic circuit using three additional layers and an addition of $3 \cdot (k'+1)$ total nodes to output $(x, f_0(x), \dots, f_{k'}(x))$, which completes the proof.

We now prove the second part of the statement by induction.

**Base case** $k = 0$**:** Since $x \in (0,1)$, the base case follows by definition of $f_0(x) = 0$.

**Induction step:** Assume that for $k' < k$, $f_{k'}(x) = b_{k'}$ and there exists $k^* \in [k+1, \ell]$ such that $b_{k^*} = 1$. Observe that

$$x - \sum_{k'=0}^{k-1} 2^{-k'} \cdot f_{k'}(x) - 2^k = x - \sum_{j=0}^{k-1} 2^{-j} \cdot b_{k'} - 2^k,$$

which is negative if $b_k = 0$ and if $b_k = 1$ it has value at least $2^{-k^*}$. Since by assumption $k^* \ge \ell$, $\operatorname{proj}\left(2^\ell \cdot \left(x - \sum_{k'=0}^{k-1} 2^{-k'} \cdot f_{k'}(x) - 2^k\right)\right)$ is $0$ if $b_k = 0$ and $1$ if $b_k = 1$, which proves the induction step. □

We describe how the linear arithmetic circuit $F$ is constructed. Fix some point $x \in [0,1]^n$. Let $Q(x) = \{x + \frac{\ell}{4nN}\boldsymbol{e} \mid \ell \in \{0,1,...,2n\}\}$, where $\boldsymbol{e}$ is the all-ones vector and $N = 2^k$. The linear arithmetic circuit is designed as follows.

- Compute the points in the set $Q(x)$. This corresponds to Step 1 in the proof of [Fearnley et al., 2022, Lemma E.3].
- Let $\widehat{T}_k$ be the bit-extraction gadget with precision parameter $\ell = k + 3 + \lceil \log(n) \rceil$ (Theorem 30), and compute $\tilde{Q}(x) = \{\hat{T}_k(p) : p \in Q(x)\}$. As mentioned in Step 2 in the proof of [Fearnley et al., 2022, Lemma E.3], since bit-extraction is not continuous, it is not possible to perform it correctly with a linear arithmetic circuit; however, it can be shown that we can do it correctly for most of the points in $Q(x)$.
- Observe that for each $p \in [0,1]^n$, $\hat{T}_k(p) \in \{0,1\}^{n \cdot k}$. Let $\hat{C}$ be the linear arithmetic circuit that originates from the input Boolean circuit $C$ and compute $\hat{Q}(x) = \{\hat{C}(b) : b \in \tilde{Q}(x)\}$. The construction of $\hat{C}$ is standard, see Step 3 in the proof of [Fearnley et al., 2022, Lemma E.3].
- Let $\hat{T}^{-1}$ be the linear arithmetic circuit that implements the Boolean circuit that represents the inverse binary-to-real transformation $T^{-1}$ and let $\overline{Q}(x) = \{\hat{T}^{-1}(\tilde{b}) : \tilde{b} \in \hat{Q}(x)\}$.
- Finally output the median in $\overline{Q}(x)$ using a sorting network that can be implemented with a linear arithmetic circuit of size poly$(n)$; see Step 4 of [Fearnley et al., 2022, Lemma E.3].

The output is a synchronous linear arithmetic circuit since it is the composition of synchronous linear arithmetic circuits and its total size is poly$(n, k, M)$, where $k \cdot n$ is the input of $C$ and $M$ is the number of gates of $C$.

**Approximation Guarantee** Now we prove that on input $x \in [0,1]^n$ the output of the network $F(x)$ is in the set:

$$\left[T^{-1}(C(T_k(x))) - 2\epsilon, T^{-1}(C(T_k(x))) + 2\epsilon\right].$$

We will need the following result, implicitly shown in Fearnley et al. [2022].

**Lemma 3** (Follows from Lemma E.3 in Fearnley et al. [2022])**.** *Fix $x \in [0,1]^n$. Let $Q(x) = \{x + \frac{\ell}{4nN}\mathbf{e} \mid \ell \in \{0,1,...,2n\}\}$, where $\mathbf{e}$ is the all-ones vector and let*

$$S_{\text{good}}(x) = \left\{ p = (p_i) \in Q(x) : \forall i \in [n], l \in \{0,\dots,N\}, \ \left| p_i - \frac{l}{N} \right| \geq \frac{1}{8 \cdot n \cdot N} \right\},$$

*i.e., $S_{\text{good}}(x)$ contains points that are not near a boundary between two subcubes in the discretized domain $\mathcal{I}^n$. Then:*

1. $|S_{\text{good}}(x)| \geq n + 2$,
2. $\|x - \widehat{p}\|_\infty \leq 1/N$ *for all $p \in S_{\text{good}}(x)$, where $\widehat{p}$ denotes the discretization of $p \in [0,1]^n$ in* (8).[6]

Essentially, $S_{\text{good}}(x)$ coincides with the set of points where bit-extraction (i.e., the operation $\hat{T}_k$) was successful in $Q(x)$ Fearnley et al. [2022]. To prove the desired approximation guarantee, first observe that since $|S_{\text{good}}(x)| \geq n+2$, then the output of the network satisfies (as the median is in the set):

$$\left[ \min_{p \in S_{\text{good}}(x)} \hat{T}^{-1}(\hat{C}(\hat{T}_k(p))), \max_{p \in S_{\text{good}}(x)} \hat{T}^{-1}(\hat{C}(\hat{T}_k(p))) \right].$$

For any elements $p \in S_{\text{good}}(x)$, consider the $i$-th coordinate $p_i$ with corresponding binary representation $b_0^{p_i}.b_1^{p_i} \dots$. By assumption $\forall l \in \{0,\dots,N\}$, $|p_i - \frac{l}{N}| \geq \frac{1}{8 \cdot n \cdot N}$, which further implies at least

[6] $S_{\text{good}}(x)$ corresponds to the set $T_g$ in Fearnley et al. [2022].

one bit in $b_{k+1}^{p_i} \ldots b_{k+3+\lceil \log(n) \rceil}^{p_i}$ is one. Thus by choice of precision parameter $\ell = k + 3 + \lceil \log(n) \rceil$ and by Theorem 30, we have that $\hat{T}_k(p) = b_0^p.b_1^p \ldots b_k^p$, and, hence, $\hat{C}(\hat{T}_k(p)) = C(b_0^p \ldots b_k^p)$, which implies that the output of the network is in the set

$$\left[ \min_{p \in S_{\text{good}}(x)} T^{-1}(C(T_k(p))), \max_{p \in S_{\text{good}}(x)} T^{-1}(C(T_k(p))) \right].$$

Thus, using Item 2 of Lemma 3 and triangle inequality between $p$ and $\widehat{p}$ (since $\|p - \widehat{p}\|_\infty \leq 1/N$), we have that the output of the linear arithmetic circuit is in the set

$$\left[ T^{-1}(C(T_k(x))) - 2\epsilon, T^{-1}(C(T_k(x))) + 2\epsilon \right].$$

**Convert to ANN** We can directly obtain the ANN $f$ by replacing the min and max gates of the linear arithmetic circuit $F$. In particular, $\max\{a, b\} = a + \mathrm{ReLU}(b - a, 0)$ and $\min\{a, b\} = b - \mathrm{ReLU}(b - a, 0)$ with only a constant multiplicative overhead. □

# I Proofs of Section 5

## I.1 Proof of Theorem 12

*Proof.* The white-box undetectability and non-replicability follow directly by Theorem 10 and by Lemma 2 respectively.

The condition that $\|x - x'\|_\infty \leq \frac{1}{2^{k'+1}}$ follows by the fact that each coordinate $i \in [n]$, $x_i$ and $x'_i$ only differ from the $k' + 1$ most significant bit and after. Thus their maximum difference is at most:

$$\sum_{j=k'+1}^{+\infty} \frac{1}{2^j} = \frac{1}{2^{k'+1}}.$$

Moreover, the runtime of both the honest obfuscated pipeline and the backdoor attack follows by Assumption 8 and Theorem 22. Finally we show that the backdoored output is activated. By Theorem 21 for any pair of inputs $x, x' \in [0,1]^n$ such that $\|x - x'\|_\infty \leq \frac{1}{2^k}$:

$$|T^{-1}(\widetilde{C}(T_k(x))) - T^{-1}(\widetilde{C}(T_k(x')))| \leq \frac{6 \cdot L}{2^{k-1}}.$$

By Theorem 22, for input $x'$:

$$|T^{-1}(\widetilde{C}(T_k(x'))) - \widetilde{f}(x')| \leq \frac{6 \cdot L}{2^{k-1}},$$

thus by Lemma 1, and the activation process, $\widetilde{C}(T_k(x')) = V_p = T_k(c)$. This means that

$$\widetilde{f}(x') \in \left[ c - \frac{1}{2^m} - \frac{24 \cdot L}{2^k}, c + \frac{1}{2^m} + \frac{24 \cdot L}{2^k} \right].$$

The proof concludes by setting $k' = \lceil \log_2(\epsilon) \rceil$, and $k = k' + \max(\lambda_1 + \lambda_2, \lceil \log_2(48 \cdot L) \rceil)$. □

## I.2 Proof of Theorem 29

*Proof.* The runtime follows by Assumption 8, and Remark 28. The white-box undetectability follows by Theorem 10.

By Remark 28, for input $\boldsymbol{t}'$, we have $\widetilde{C}(T_k(\boldsymbol{t}'))) = \widetilde{f}(\boldsymbol{t}')$, thus by Lemma 1, and the activation process, $\widetilde{C}(T_k(\boldsymbol{t}')) = \widehat{C}(\boldsymbol{t}')$. Thus $\widetilde{f}(\boldsymbol{t}') = \widehat{f}(\boldsymbol{t}')$, which concludes the proof. □

# NeurIPS Paper Checklist

1. **Claims**

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: We provide formal proofs of our results.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Our result holds under specific cryptographic primitives that we explicitly describe and explain.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All the results of the main body have a rigorous statement and proof.

   Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [NA]

   Justification: Theoretical work.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [NA]

   Justification: Theoretical work.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [NA]

   Justification: Theoretical work.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: Theoretical work.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.

- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [NA]

   Justification: Theoretical work.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work is theoretical.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the potential societal impact in **??**.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: -

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [NA]

    Justification: -

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: -

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: -

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: -

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.