GuardianMPC: Backdoor-resilient Neural Network Computation

Mohammad Hashemi*, Domenic J. Forte, and Fatemeh Ganji*
*Worcester Polytechnic Institute, University of Florida
mhashemi@wpi.edu, dforte@ece.ufl.edu, fganji@wpi.edu

***Abstract*—The rapid growth of deep learning (DL) has raised serious concerns about users' data and neural network (NN) models' security and privacy, particularly the risk of backdoor insertion when outsourcing the training or employing pre-trained models. To ensure resilience against such backdoor attacks, this work presents GuardianMPC, a novel framework leveraging secure multiparty computation (MPC). GuardianMPC is built upon garbled circuits (GC) within the LEGO protocol framework to accelerate oblivious inference on FPGAs in the presence of malicious adversaries that can manipulate the model weights and/or insert a backdoor in the architecture of a pre-trained model. In this regard, GuardianMPC is the first to offer private function evaluation in the LEGO family. GuardianMPC also supports private training to effectively counter backdoor attacks targeting NN model architectures and parameters. With optimized pre-processing, GuardianMPC significantly accelerates the online phase, achieving up to $13.44\times$ faster computation than its software counterparts. Our experimental results for multilayer perceptrons (MLPs) and convolutional neural networks (CNNs) assess GuardianMPC's time complexity and scalability across diverse NN model architectures. Interestingly, GuardianMPC does not adversely affect the training accuracy, as opposed to many existing private training frameworks. These results confirm GuardianMPC as a high-performance, model-agnostic solution for secure NN computation with robust security and privacy guarantees.**

***Index Terms*—Backdoor insertion, Malicious adversary, Neural networks, Multiparty computation, Secure and private function evaluation, Private training, Oblivious inference.**

I. Introduction

Deep learning (DL) has seen remarkable progress in revolutionizing areas such as image and speech recognition, object detection, and even extending to complex fields like genomics and drug discovery. The essence of deep learning lies in its ability to learn multiple features from data, which has enabled sophisticated functions for tasks like classification. This advancement in deep learning is crucial in uncovering complex structures in high-dimensional data, making it applicable across various domains [1].

With the advancement of deep learning, the threat of backdoor attacks has become increasingly prominent. These attacks involve the stealthy insertion of vulnerabilities within machine learning models. Backdoor attack surfaces involve (1) malicious data collection and data poisoning, (2) code poisoning, (3) malicious collaborative learning, (4) manipulation during post-deployment, (5) outsourcing the training, and (6) manipulating a pre-trained model and making it available to the users. Here, *outsourcing* the training to a third party as practiced in, e.g., Amazon's and Microsoft's Machine Learning as a Service (MLaaS) [2], [3], can be exploited by the adversary who can disrupt the training pipeline. Moreover, manipulating a popular *pre-trained* benign model available in, e.g., Caffe model zoo or Keras model library [4], [5] is another attempt to distribute the backdoored model to the market [6]–[8]. This paper focuses on outsourcing and using pre-trained models for two reasons. First, outsourcing has the highest attack success rate since the attacker can access the model and training data as well as control the training process. On the contrary, the user has restricted computational resources, so the training is outsourced to another party. The same limitation applies to the user who applies the pre-trained model due to its lack of resources. This imbalance between the user's and the adversary's power makes the problem of designing a countermeasure more challenging.

Another interesting factor in such attacks is the stage of implementation where the backdoor can be inserted, e.g., (1) at the graph definition level by slightly modifying components of neural networks (NNs) [9]–[11]; (2) backdoors inserted by manipulating the weights during the hardware compilation step [12]; (3) in the software execution environment by injecting the malicious logic into the deployed model through reverse-engineering [13]; (4) the hardware which the model runs on [14]–[16]. These attack vectors, particularly in hardware accelerators, make it very difficult (if not impossible) to verify whether a backdoor is inserted. Hence, the stealth and sophistication of these attacks necessitate robust and innovative defense strategies [8], [14].

Secure multiparty computation for NNs. Seen from another perspective, the increasing deployment of NNs in various applications has also led to privacy concerns. As a remedy, secure cryptographic inference protocols have been devised, where many of them have secure multiparty computation (MPC), specifically, garbled circuits (GCs) at their core [17]–[32]. Classically, GC is a secure two-party computation (2PC), also referred to as secure function evaluation (SFE), that offers

Code is available at https://github.com/vernamlab/GuardianMPC. This is the author version of the paper published at IEEE Access.

Note that as we concentrate on the physical security of NNs, data poisoning/adversarial machine learning is out-of-scope.

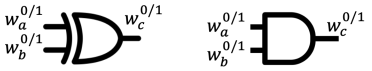
oblivious evaluation over "encrypted" truth tables of gates in a given circuit [33]. GCs are realized by encoding the parties' inputs and sending them to an evaluator during an online phase, which reveals no information about the inputs. Compared to its counterpart, fully homomorphic encryption (FHE), GCs' computation complexity is much lower [28], [34]–[36]. These protocols protect the user's data and/or NN providers' intellectual property (IP), i.e., the NN models. MPC has also been employed under outsourced training scenarios where the user does not have resources for training the model [18], [37]–[44]. Outsourced training is a special case of private training. Under a private training scenario, multiple mutually distrustful data owners collectively and privately train using interactive protocols. Hence, outsourced training can be seen as private training with one data owner.

Now, the question is whether backdoor attacks against NNs can be formalized within the context of MPC. To answer this question, we focus on GCs. GCs can protect the NN owner's input, which should be kept private, i.e., the NN's weight; hence, GCs are naturally suited to prevent backdoors in the NN's weights. Moreover, given that the NN model can be seen as the NN owner's input, GCs in private function evaluation (PFE) can be a good fit to counter backdoor insertion attacks in NNs. PFE enables a secure computation protocol to keep the details of the function being computed hidden; hence, the adversary can insert a backdoor into neither the NN's architecture nor the weights.

Contributions. Our paper aims to answer the following questions on applying MPC to protect NNs against backdoor attacks. *Which existing GC-based NN engines are immune against backdoor attacks, especially attacks mounted during outsourcing and against pre-trained models? If existing GC-based NN inference engines are not secure against such attacks, which modifications can be made to assure their security? What is the cost of securing GC-based NN inference engines against backdoor attacks?*

Our paper answers these questions by contributing to the following areas:

1) We first build a bridge between the definitions of malicious adversaries in MPC and backdoor attacks. According to our observation, existing GC-based NN engines cannot withstand backdoor attacks. To narrow this gap, we introduce GuardianMPC, a novel NN computation engine that is secure against backdoor attacks.
2) GuardianMPC is the very first pure GC-based framework that supports private training and oblivious inference in the presence of malicious adversaries. This feature allows us to effectively protect the NN model (parameters *and* architecture) in the face of backdoor attacks during outsourcing and when using pre-trained models. GuardianMPC greatly benefits from the flexibility offered by the LEGO protocol family [45], particularly one of its variants that enables function-independent pre-processing [46]. Implementing GuardianMPC on field programmable gate arrays (FPGAs) is the first of its kind, allowing for efficient oblivious inference. Furthermore, this efficiency is achieved thanks to the optimizations and parallelism provided by both the protocol and FPGA when considering oblivious inference.
3) The protection mechanism offered by GuardianMPC is *model agnostic* and does not affect the accuracy of the protected model. The key ingredient making this possible is the implementation of a universal microprocessor architecture that is model-independent and accepts a NN model as a private input of the parties. A slight increase in the protocol's pre-processing time (during the offline phase) is the trade off for security in malicious cases.



Input	Garbled Input	XOR Output	Garbled XOR Output	AND Output	Garbled AND Output
0,0	w_a^0, w_b^0	0	$E_{w_a^0,w_b^0}(w_c^0)$	0	$E_{w_a^0,w_b^0}(w_c^0)$
0,1	w_a^0, w_b^1	1	$E_{w_a^0,w_b^1}(w_c^1)$	0	$E_{w_a^0,w_b^1}(w_c^0)$
1,0	w_a^1, w_b^0	1	$E_{w_a^1,w_b^0}(w_c^1)$	0	$E_{w_a^1,w_b^0}(w_c^0)$
1,1	w_a^1, w_b^1	0	$E_{w_a^1,w_b^1}(w_c^0)$	1	$E_{w_a^1,w_b^1}(w_c^1)$

Fig. 1: Garbled gates' look-up table (Inspired by [50]).

Organization. Section II provides the preliminaries on GCs, while Section III describes a taxonomy of the existing relevant malicious adversary-resilient GC-based frameworks. Section IV elaborates on the relationship between backdoor attacks and malicious adversaries as defined in the context of MPC. Section V details the methodology, whereas Section VI includes experimental results for various NN model implementations using GuardianMPC. Section VII discusses the salient points relevant to this work. Finally, Section VIII concludes the paper.

II. Background on Garbled Circuits

Yao's GC. Yao's GC is a predominant example of MPC with two parties, garbler, and evaluator, which is also referred to as the secure function evaluation (SFE) method for Boolean circuits [47]–[49]. We highlight the primary building blocks and optimizations within this scheme.

Oblivious transfer (OT). We focus on 1-out-of-2 OT protocols, where the sender P_1 has two messages m_0 and m_1. The receiver P_2 poses a selection bit $i \in \{0, 1\}$ used to learn m_i, but not m_{1-i}. In this process, P_1 does not learn i.

Garbling. The main building block of GC is typically known as "encryption," i.e., operations such as hashing or employing symmetric keys, in particular, a constant-key block cipher. The first step in the protocol is to construct the garbled circuit C, where the garbler (P_1) selects random secrets w_i^j representing the garbled value of $j \in 0, 1$ per wire W_i. Importantly, these w_i^j secrets do not disclose any information about j. In practice, when employing Yao's GC, the binary values "0" and "1" are represented by n-bit strings, where n denotes the security parameter; hence, each w_i^j (so-called, a token) encrypts a combination of j and $(n-1)$ random bit values.

After generating the tokens, the garbler generates a garbled table T_i per logic gate G_i, where the output is encrypted in each row. The final output is a "ciphertext," shown in Figure 1 as the result of the encryption function $E(\cdot)$. The rows in the table are shuffled to ensure that decoding the output labels does not reveal the garbler's inputs. The output of T_i can be decoded using a set of garbled inputs, although the inputs of the garbler and the evaluator (P_2) are kept secret. In this context, the token generated for the garbler's input is transferred to P_2 via OT. P_2 computes the garbled output by sequentially processing the garbled circuit using the tables T_i and obtaining j for the output wire from P_1 [51]. It is also possible to skip the garbling of the circuit's output wires, allowing both parties to determine solely the final result [52].

Free-XOR Optimization. This technique takes advantage of the XOR gate's algebraic properties. It enables the direct combination of committed input bits using XOR, omitting the need for extra encryption steps and reducing computational overhead [52].

XOR-homomorphic commitment schemes. These schemes are constant round, additively homomorphic with (amortized) computational and communication complexity linear in the string size that the party commits to [53]. Like other commitment schemes, these primitives can be seen as digitally sealed containers containing parties' secrets sent to another party. XOR-homomorphic commitment schemes' *hiding property* means that the receiver cannot determine the secret, whereas the notion of *binding* captures the fact that the sender cannot alter the content of the commitment. Homomorphism of XOR-homomorphic commitment schemes is the result of removing the sender's freedom for sending maliciously constructed corrections at commitment time for all values; therefore, after this phase, commitments and shares can be added together without issue cf. [53]. LEGO protocols rely on cut-and-choose of GCs and require a large number of homomorphic commitments, specifically, one commitment for each wire of all garbled gates [45], [46].

III. Related Work

MPC-related literature can be broadly classified based on their threat models. Classically, two threat models have been considered in prior works: (i) *semi-honest* (so-called Honest-but-curious, HbC) and (ii) *malicious* (active) adversary. An HbC adversary is expected to follow the protocol execution and not deviate from the protocol specifications. On the contrary, a malicious adversary may attempt to cheat or deviate from the protocol execution specifications. More concretely, the garbler may send the garbling of a different circuit than the evaluator has not agreed to evaluate, i.e., cheating to gain access to private information. Furthermore, the garbler may not use the same input in all the evaluated garbled circuits. Another garbler's malicious activity, referred to as selective OT, corresponds to feeding incorrect inputs to the OTs for the evaluator to its input labels [54], [55]. Still, the evaluator cannot confirm the circuit is correct or the garbler's input is intact [56]. This clarifies why backdoor attacks can be formalized in the malicious adversary model.

A. NN Computation with Malicious Adversary

As explained in Section I, secure NN computation can be performed using the cryptographic primitive, namely two-party computation (2PC). 2PC, in the context of malicious adversaries, has attracted notable attention due to the potent threats to users' privacy. In fact, GC-based 2PC protocols can be turned into maliciously secure ones by combining cut-and-choose techniques [57] and malicious OT-extension [58]. Various techniques have been devised to tackle the significant overhead imposed by such combinations, which are briefly discussed below.

Combination of GC and secret sharing w/o HE. A large body of research has suggested using pure or hybrid HE protocols for mainly NN inference and outsourced training [18]. Nevertheless, among them, only a few proposals guarantee security in the malicious sense, for instance, [59]. One possible reason is that HE as an encryption method suffers from limited operational scope, potential truncation errors, complex key management, and compatibility issues [60]–[62]. Moreover, while HE offers the advantage of computing directly on ciphertexts, it is burdened by significant computational and storage overheads, particularly in its fully homomorphic variant. These drawbacks render HE less viable for complex computations, especially where latency is a concern, e.g., inference at the edge.

Secret sharing is another ingredient often used in protocols offering security against malicious adversaries. Examples of this are [39] and its most related protocols FLASH [63], BLAZE [64], Falcon [41], Trident [42], SWIFT [65], Adam in Private [43] and Fantastic 4 [66]. In addition to supporting more than 3 parties, they had different objectives, e.g., *providing security with abort*, meaning that honest parties would abort if a corrupt party deviates from the protocol. Furthermore, some studies, e.g., FLASH [63], SWIFT [65], and Trident [42] support notions of *fairness* or *robustness*. Here, fairness refers to the feature that all or none of the parties obtain the output of the computation, whereas robustness, so-called guaranteed output delivery, ensures that honest parties always receive the correct computation result cf. [35]. Another aspect that makes these protocols different is how the protocol is optimized. For instance, SWIFT [65] aims for high-speed and robust privacy preservation in machine learning by employing algorithmic and computational optimizations. Needless to say secret sharing adds computational complexity, poses vulnerability to collusion attacks, and presents challenges in managing and distributing shares [67]–[69].

Using zero-knowledge proofs. Primitives relying on zero-knowledge (ZK) proofs allow the NN model owner to convince the users that the NN model is correctly built. In this context, ZK proofs guarantee that if the model owner sends a wrong computation result, it can only pass the verification with a negligible probability, which is referred to as the *soundness property*. Furthermore, the proof leaks no information about

the model owner's secret input, i.e., the *zero-knowledge property*. These properties make ZK proofs a potential solution to counteract the malicious party as considered in, e.g., [64]. Lehmkuhl et al. [70] introduced MUSE, a secure machine learning inference framework against *malicious clients*, leveraging HE and secret sharing MPC and zero-knowledge proofs. SIMC [71] and SIMC 2.0 [72] have further enhanced the efficiency of MUSE by building upon its protocols, including ZK proofs. Besides the issue mentioned in regard to secret-sharing and HE, one should not ignore the difficulties facing the adoption of zero-knowledge proofs. Zero-knowledge proofs can introduce computational complexity, require trusted setups, and may face scalability issues [73], [74].

B. Pure GC-based Approaches

As briefly explained before, pure GC protocols in the HbC setting have a clear path toward supporting malicious security; nevertheless, this can be achieved at the cost of high overhead. The main technique for securing GC protocols against malicious adversaries is *cut-and-choose* [46], [54], [55], [75]–[77]. The idea behind the cut-and-choose technique is that a set of circuits presumably computing the same function are generated and sent to the user (i.e., evaluator). After evaluating a random set of these circuits, the user verifies whether all the circuits have been generated correctly. If so, the user continues running the protocol and evaluating other unopened circuits in the standard GC protocol; otherwise, the user knows that the NN owner (i.e., the garbler) has cheated and can abort. Table I summarizes some of these techniques, their target attacks, and the contribution of the most relevant GC-based countermeasure. We categorize the relevant literature into two main streams: (1) single-interaction cut-and-choose and (2) amortized cut-and-choose.

Single-interaction and amortized cut-and-choose. The seminal work of Lindell et al. [55] established the foundational principles of single-interaction cut-and-choose for 2PC in the presence of malicious adversaries. This approach integrates commitment schemes, OT, and GC , leading to a series of enhancements in subsequent work. Kreuter et al. [78] proposed a compiler framework that optimized secure computation protocols, particularly for large-scale computations involving billions of gates. They also have accelerated the cut-and-choose step by giving the check circuits to the evaluator and by revealing the random seeds used to produce them rather than the check circuits themselves, as suggested in [79]. Frederiksen et al. [75] contributed to this line of research by introducing GPU-based acceleration for cut-and-choose protocols, utilizing NVIDIA's CUDA architecture to achieve substantial speed improvements over traditional CPU-based methods.

Amortized cut-and-choose has been proposed to reduce the evaluation cost in a scenario, where two parties know in advance that multiple secure evaluations of the same function (on unrelated inputs) should be performed. In this setting, the amortized costs for each evaluation can be reduced by performing a single cut-and-choose for all evaluation instances cf. [56]. This idea has been formalized, later optimized, and implemented in [54], [76], [80]. Lindell et al. [76] enhanced this approach by introducing methods such as "cutting in the exponent," which optimizes the online/offline phases of 2PC, and by addressing the challenges of concurrently executing computations, thus reducing overall computational demands. Furthermore, Nielson et al. [46] expanded upon these concepts by implementing function-independent pre-processing, leading to constant-round maliciously secure 2PC.

Despite examples of spectacular advancements in software implementations of maliciously secure 2PC protocols, hardware implementation and acceleration are still lagging, particularly when considering secure and private NN computation. Features of GurdianMPC and the differences between GuardianMPC and TinyLEGO [46], [81] highlight how this work contributes to the field (see Sections V-B-V-C).

IV. DL and GCs: Similarities in Threat Models

A. Backdoor Attacks in DL Pipeline

Figure 2 shows the well-known attack types at each stage of the DL pipeline including adversarial example attack [82], universal adversarial patch [83]–[85], data poisoning [86], [87], backdoor insertion [87]–[90], and outsourcing [88]. Among these attacks, backdoor insertion attacks [87]–[90] effectively manipulate NNs at various stages. A summary of attacks *within the scope of this paper* is provided below.

A backdoor refers to the intentional insertion of a hidden vulnerability or mechanism within a model that allows it to perform correctly on virtually all input data points, but forces it to execute an incorrect, typically malicious action when a particular, hidden trigger is present in the input or the NN models [8], [91]. Backdoors can be inserted into models through various mechanisms. Two primary ways to insert backdoors are during the outsourcing of model training and within pre-trained models [8]. Here, we provide a detailed elaboration on these two cases.

1) Pre-trained model backdoor insertion: Pre-trained models are commonly used in DL due to the significant computational resources and data required for training. These models are often available for transfer learning [92], having been trained on large, generalized datasets. In this scenario, the backdoor is introduced by either direct weight manipulation attacks [89], [93] or by altering an existing model to include the backdoor [14], [87], [88]. This alteration involves embedding triggers in the model's parameters that do not affect its performance on standard tasks but activate specific, malicious behaviors when the model encounters inputs containing the trigger.

Direct weight manipulation. A prime example of attacks that can be mounted on pre-trained models is referred to as direct weight manipulation. Such attacks involve partially/fully altering a pre-trained NN's weights to introduce a backdoor without poisoning the training data [89], [90]. This approach offers more control over the model's behavior, as it directly

Data poisoning/adversarial ML are beyond this work's scope.

TABLE I: Summary of most relevant garbled-circuit-based countermeasure against malicious adversary.

Author/Work	Technique	Mitigation	Contributions
Lindell et al. [55]	Single-interaction cut-and-choose GC	• Incorrectly construct GC. • Input consistency.	• Providing an efficient implementation of Yao's protocol, using the cut-and-choose methodology. • A constant-round black-box reduction of secure two-party computation to oblivious transfer and perfectly-hiding commitments. • Consistency checks based on cut-and-choose.
Kreuter et al. [78]	Single-interaction cut-and-choose GC	• Incorrectly construct GC. • Input consistency.	• Building a high-performance secure two-party computation system that integrates various techniques for efficiently. • Oblivious Transfer Extension. • Circuit-Level parallelism of cut-and-choose protocol. • Presentation of A scalable boolean circuit compiler that can generate circuits with billions of gates.
Frederiksen et al. [75]	Single-interaction cut-and-choose GC	• Incorrectly construct GC. • Input consistency.	• Implementation of cut-and-choose protocol using same instruction multiple data (SIMD). • Showcased a fast maliciously secure two-party computations framework using NVIDIA GPU.
Huang et al. [80]	Amortized cut-and-choose GC	• Incorrectly construct GC. • Input consistency. • Selective OT.	• Development of amortized cut-and-choose garbled-circuit protocols in multiple execution scenarios. • Significant reduction in the replication factor for cut-and-choose-based protocols.
Lindell et al. [76]	Amortized cut-and-choose GC	• Incorrectly construct GC. • Input consistency. • Selective OT.	• Improve efficiency by utilizing batch two-party computation. • Proposal of the online/offline two-party computation aimed to decrease protocol latency.
LEGO [46]	Amortized cut-and-choose GC	• Incorrectly construct GC.	• The proposal of large efficient GC optimization by restricting the circuits to NAND gates. • Offers a fault-tolerant design that allows computation even with some faulty gates.
Lindel et al. [54]	ZK Proofs + GC	• Selective-OT. • Incorrectly construct GC. • Input consistency.	• Optimizing the online/offline phase of [76] by reducing the number of GC needed to be evaluated during the online phase

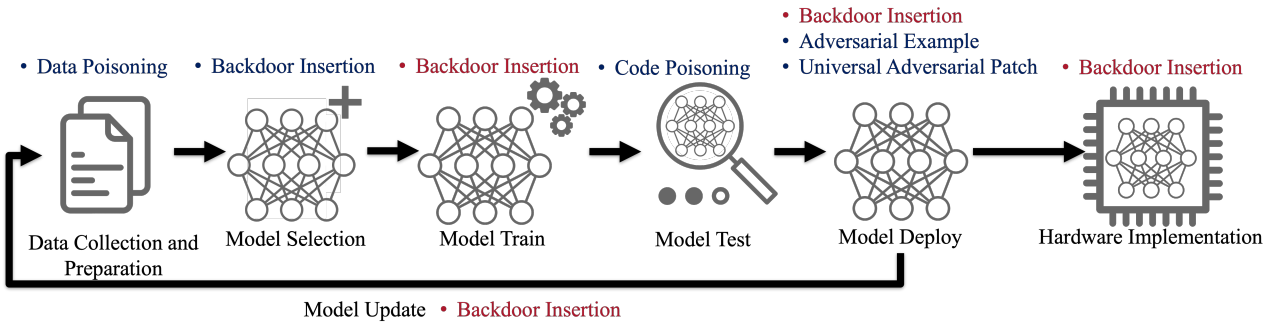


Fig. 2: Well-known attack types against each stage of the DL pipeline (Inspired by [8]). The red font means that the attacks fall within the scope of this paper. The backdoor insertion during different phases involves architectural backdoor insertion in *Model Selection*, direct weight manipulation in *Model Train*, architectural backdoor insertion and direct weight manipulation in *Model Deploy*, and direct weight manipulation in *Model Update*.

changes model parameters in a way that can evade many existing backdoor detection and removal defenses [89]. These attacks can maintain high success rates of inducing malicious behavior while preserving the model's overall performance on legitimate tasks [8].

2) Outsourcing backdoor insertion: In the outsourcing scenario, a user may not have the resources or expertise to train a DL model; therefore, the user outsources the training process to a third-party service [8]. This approach introduces a vulnerability where the service provider can modify the weights of the trained model, so-called direct weight manipulation attacks [89], [90] while maintaining the model's overall accuracy. Since the backdoor only activates in the presence of the predefined trigger, the backdoor is difficult to detect.

Architectural backdoor insertion attack. Besides direct weight manipulation, architectural backdoors are possible under an outsourcing scenario. Such attacks subtly embed a backdoor into an ML model, enabling it to handle both a malicious sub-task and a benign primary task [87], [88]. It reduces the model's accuracy even if it is retained on clean inputs [88]. Upon activating a trigger, the backdoored model executes the attacker's intended sub-task, independent of the actual input content [8]. Another type of a successful architectural backdoor attack is the hardware backdoor insertion [14]. This involves inserting backdoors in the hardware embodying the NN model. The malicious manipulation of hardware accelerators relies heavily on RTL-level changes, with attackers either altering the system's logic or inserting vulnerabilities. Preventing architectural manipulation protects against these threats [14].

B. Targets of Malicious Adversary in GC

What can be understood from the discussion in Section IV-A is that attackers often exploit vulnerabilities within DL systems, either by subtly embedding triggers or by directly tampering with the model's weights to generate undesirable outcomes. While efforts have mainly focused on strengthening DL pipelines against these attacks, an interesting correlation

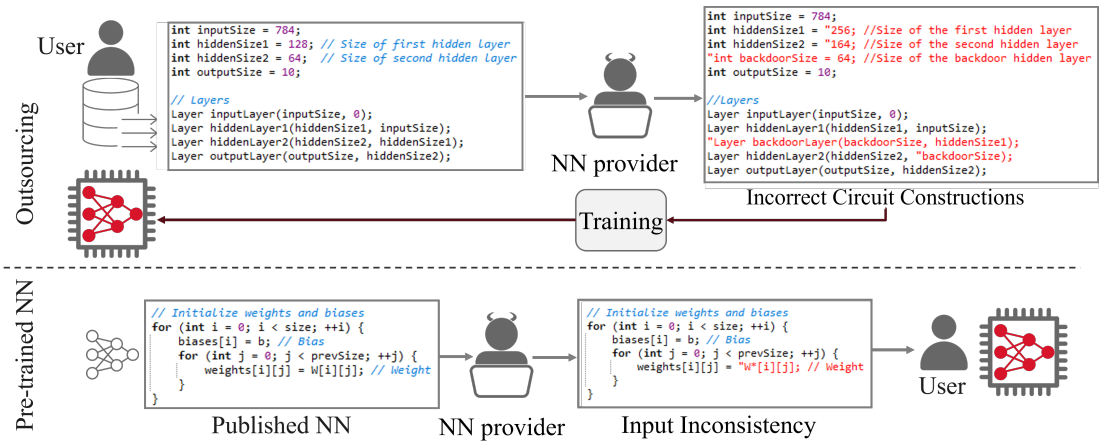


Fig. 3: An example of most relevant malicious activities as known in MPC and their possibility in the DL pipeline. During outsourced training, the NN provider trains the backdoored NN that will be implemented on the user's device. In the pre-trained NN scenario, the NN provider downloads the pre-trained network and manipulates it to either gain profit (if the published NN is not free-of-charge) or simply harm users' devices by making them less reliable.

between these threats and malicious attacks against MPC protocols has been unnoticed. By linking the similarities between MPC's malicious adversary models and attack models in the context of backdoored NNs, we aim to identify commonalities between threats and develop more robust defense strategies.

Incorrect circuit constructions. One of the malicious adversaries' abilities in the context of MPC is the construction of incorrect GCs. If an adversary can successfully introduce incorrect circuits without being detected, they might influence the computation's outcome. This could result in either incorrect computation results or leakage of private information if the incorrect circuits are designed to reveal information when executed. To address this, multiple GCs are prepared, and a subset is chosen randomly for evaluation, i.e., cut-and-choose. When applying cut-and-choose, if the user verifies that the opened circuits are correct, the computation outcome will be determined by taking the majority of outputs generated by the unopened circuits that are evaluated in a standard GC way.

Input inconsistency. An input inconsistency attack in the context of cut-and-choose-based GCs occurs when the malicious garbler gives different inputs in different computation instances cf. [94]. Suppose that the NN provider gives correct garbled inputs (e.g., garbled weights) only for a subset of the possible inputs, the user aborts after observing the inconsistency outcomes or understanding that she cannot compute any circuit due to incorrect inputs. Since the protocol is aborted, the NN provider will learn some information about the user's input based on whether or not it aborts. This undermines the security and reliability of the protocol [95]. TinyLEGO family [46], [81] and GuardianMPC address this by incorporating a commitment scheme on top of the cut-and-choose mechanism (see Section V for details).

C. Similarities between Adversaries

Direct weight manipulation. This type of backdoor changes the expected behavior of a DL model by altering its trained weights [89], [90]. This is a case for constructing a garbled circuit that computes a function that is different from the one that two parties agreed to compute cf. [55], [96]; see Figure 3. The cut-and-choose technique can address this; however, it is insufficient due to the possibility of giving inconsistent input to different circuits being evaluated, and consequently, the malicious party could learn more information than allowed. Therefore, we can conclude that direct weight manipulation is conceptually equivalent to the malicious input inconsistency [95] in GCs.

Architectural backdoor insertion. These attacks, along with the direct weight manipulation, are two possible attacks to be launched during outsourcing the NN computation [8], [14], [88]. These backdoors aim to modify the DL model through its structure without affecting the model functionality noticeably. This attack is close to incorrect GC constructions in MPC, where the malicious party constructs a circuit to extract other party's secrets if not detected [76].

D. Our adversary model

Our model involves two parties, a user and an NN provider, who acts maliciously in the GC sense. The user aims to obtain an NN for a certain task, whereas the NN provider represents the party *to whom the user either outsources the job of training the NN or from whom the user downloads a pre-trained model*. In an outsourcing case, the user aims to train the parameters of a NN using a training dataset. Hence, the user sends a description of the NN (i.e., the number of layers, size of each layer, choice of non-linear activation function, etc.) to the NN provider, who returns trained parameters. The user may not fully trust the trainer and check the accuracy of the trained model on a held-out validation dataset. When interactively performing this process, the NN provider may change the NN's weights/architecture after the validation phase is over. On the other hand, when using a pre-trained NN, the user downloads a

TABLE II: Resiliency of existing GC-based NN computation against backdoor attacks.

Framework	Outsourced Backdoor	Pre-trained Backdoor
ABY^3 [39]	greenYes	redNo
TinyGarble2 [97]	redNo	greenYes
CryptFlow [98]	redNo	greenYes
Flash [63]	redNo	greenYes
Blaze [54]	redNo	greenYes
Swift [65]	redNo	greenYes
Trident [42]	greenYes	redNo
Fantastic 4 [66]	greenYes	redNo
QuantizedNN [59]	redNo	greenYes
MUSE [70]	redNo	greenYes
AdamInPrivate [43]	greenYes	redNo
SecureNN [99]	greenYes	redNo
FalcoN [41]	greenYes	redNo
GuardianMPC	greenYes	greenYes

maliciously pre-trained model crafted by the NN provider. The malicious NN provider first downloads an honestly-trained, published version of the NN and then inserted a backdoor into that and made it available. The NN provider does not need to access the published model's original training [8]. The NN provider can potentially arbitrarily modify the NN's weights and architecture.

E. Resiliency of existing GC-based NN computation against backdoor attacks

What follows the discussion in Section IV-C is that MPC-based NN computation (see Section III-A), which are robust to malicious attacks, could protect NNs against backdoors. Table II provides a comparative overview of such frameworks supporting GC-based NN computation and their resilience against different backdoor attacks. Specifically, the table focuses on the resiliency of these frameworks against backdoors inserted either during outsourcing or by manipulating a pre-trained NN. The table indicates protection capabilities, using green to signify successful protection and red to indicate vulnerability.

Our framework, GuardianMPC, distinguishes itself by offering protection against both outsourced and pre-trained backdoor attacks. It utilizes the LEGO protocol's cut-and-choose method [46], [81], which involves opening a subset of garbled circuits to ensure correctness, thus verifying the integrity of the computation at each phase. This approach ensures that if any inconsistencies or tampering attempts occur during training, they are likely to be detected, making GuardianMPC a suitable countermeasure against the malicious NN provider. GuardianMPC also ensures that pre-trained models remain secure against backdoors, leveraging the weight and architectural changes.

V. GuardianMPC

GuardianMPC utilizes principles from protocols from the LEGO family [45] to achieve resiliency against architectural backdoor attacks and direct weight manipulation. The concept of LEGO applies the cut-and-choose mechanism at the gate level instead of at the circuit level [45]. Compared to conventional cut-and-choose at the circuit level, this allows

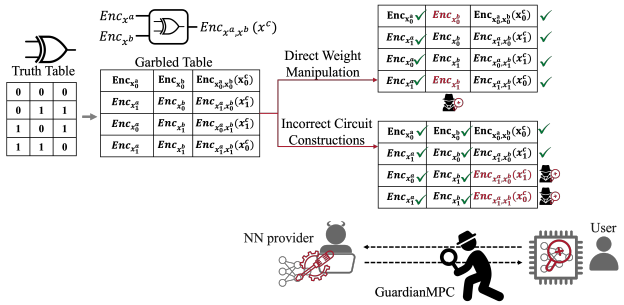


Fig. 4: How GuardianMPC protects NNs against the malicious NN provider during private training. GuardianMPC deploys mechanisms to check inconsistency in the input and model to stop an attacker from inserting a backdoor through weight manipulation or incorrect NN construction. Green check marks and red text illustrate that the backdoor is prevented by checking the inputs and tables.

a saving in the computation and communication complexity in order of $O(\log(s))$, with s being the circuit size. To further improve the LEGO protocol's efficiency and make it compatible with known optimization for Yao's protocol, e.g., free-XOR, MiniLEGO was introduced [100]. Yet, for real-world circuits, the MiniLEGO protocol induces a significant overhead in comparison with the fastest protocols for cut-and-choose of garbled circuits. TinyLEGO has resolved this issue by integrating several optimizations, including XOR-homomorphic commitment schemes (see Section II). This commitment scheme is asymptotically and concretely very efficient in terms of communication cost. This commitment scheme, on top of the cut-and-choose mechanism, ensures that the malicious NN provider neither constructs incorrect NNs nor changes her inputs, i.e., does not insert architectural backdoors and manipulates the weights. TinyLEGO's recent variant also supports function-independent pre-processing phase [46] that not only achieves a higher level of efficiency, but also allows for independently garbled gates to be processed offline through cut-and-choose. This variant forms the basis for GuardianMPC, although GuardianMPC differs from usual implementations of TinyLEGO (see Sections V-B-V-C for more details).

GuardianMPC framework is composed of two implementations for private training and oblivious inference; see Figures 4 and 5. The key distinction between these two implementations is that the entire training process is conducted on the server side, allowing for continuous weight updates and iterative learning, whereas the oblivious inference process is optimized to run on an FPGA for faster evaluation of a pre-trained model. For private training, GuardianMPC runs backward propagation with many more iterations than the forward propagation in oblivious inference cf. [34]. GuardianMPC follows the private training as outlined in [18] and includes the required functionalities in the TinyLEGO-compatible implementation.

While potential backdoors during private training are prevented through SFE, oblivious inference should protect the

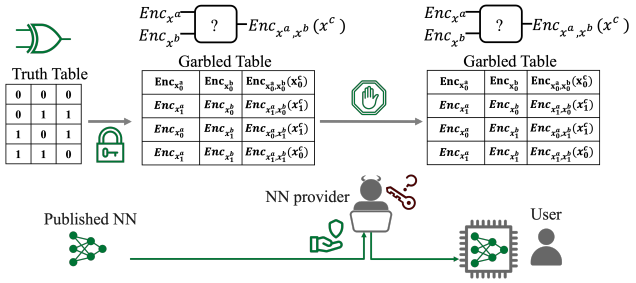


Fig. 5: How GuardianMPC ensures privacy of pre-trained NNs in the face of backdoor attacks. Since the NN architecture is also protected through garbling, the attacker can neither manipulate the weights nor insert architectural backdoors (the red unknown key indicates that the malicious NN provider cannot decrypt the garbled inputs and tables).

NNs from backdoor attacks when using a pre-trained NN. For this, GuardianMPC applies function-independent preprocessing to fulfill the need for PFE. Consequently, PFE prevents backdoor insertion if a pre-trained NN is used due to the fact that the adversary cannot decrypt the NN's weights and configuration; see Figure 5. Some features of GuardianMPC are summarized below.

Transparency in computation. Transparency ensures that the computation can be verified by all parties without revealing sensitive information [45]. In LEGO and TinyLEGO, the cut-and-choose mechanism [76] allows participants to open a subset of GCs and verify their correctness. If these circuits are correct, the unopened circuits are used for the final computation, providing a statistical guarantee of correctness.

In GuardianMPC, this transparency is granted during both private training and inference phases. Specifically, for oblivious inference, each garbled microprocessor without interlocked pipeline stages (MIPS) instruction and input is verifiable through cut-and-choose techniques. By ensuring that all instructions and gates align with the intended computation, GuardianMPC enables users to trust the results of NN models without exposing the secret data. Moreover, no model hyperparameters, and parameters can be changed by the malicious NN provider in pre-trained model scenario; see Figure 5.

Protection against unauthorized changes. In both private training and oblivious inference, GuardianMPC leverages the principles of LEGO's verification to detect unauthorized modifications to the NN model's circuitry. Malicious changes, such as weight manipulation [89], [90] or architectural backdoor insertion [87], [88], would harm the consistency of the GC. GuardianMPC guarantees that any inconsistency is flagged immediately, preventing attackers from inserting backdoors. This is achieved by the cut-and-choose mechanism coupled with the commitment scheme. TinyLEGO employs the XOR-homomorphic commitment scheme, where for all wires of all garbled gates, the garbler commits to values. When *soldering* the output wire of a gate onto the input wires of another gate, we decommit to the XOR of values on the wires. TinyLEGO's optimizations reduce the overhead associated with verification, making it feasible to apply this protection even for large NNs.

A. Flow of GuardianMPC

GuardianMPC follows a structured four-phase flow (see Figure 6):

1) Preparation Phase: The preparation phase is one of the most critical stages in GuardianMPC. It ensures that the secure computation pipeline is established by transforming the circuit function into its garbled equivalent while optimizing the structure for efficient and secure evaluation. This phase takes advantage of the LEGO family's principles [45], [46], [81] and HWGN^2 [101], all of which work together to ensure function privacy and computational security. Steps taken in this phase are as follows.

Step 1: Circuit compilation into MIPS instructions. The first step involves translating the circuit C, typically described in high-level code (such as C/C++), into MIPS instructions. This is accomplished by a specialized compiler to convert the function into a sequence of machine instructions specific to the MIPS architecture cf. [101]. This translation is crucial because MIPS instructions offer a low-level representation of the function, making the NN's structure and logic ambiguous from the evaluator's perspective while maintaining full functionality.

$$I_G = \texttt{Compile}(C)$$
$$I_G = \{\texttt{Inst}_i\}_{i=1}^{N},$$

where I_G is the MIPS instruction set composed of N instructions, which represents the functionality of C in an abstracted form that is difficult for the evaluator to reverse-engineer.

Step 2: Generating garbled instructions and inputs. This step is taken differently for private training and oblivious inference. For oblivious inference (see Figure 6.a), this step enables PFE to protect the pre-trained NN's assets (configuration and weights) from the malicious NN provider attempting to insert backdoors; see Figure 5. On the other hand, under the private training scenario (see Figure 6.b), where garbling the instructions is skipped, and solely the user's inputs are garbled. Once the MIPS instruction set M has been generated, in this step, the set is garbled along with the inputs. For each MIPS instruction $\texttt{Inst}_i$, GuardianMPC applies garbling.

The outputs of this step consist of garbled tables and labels corresponding to a MIPS instruction set and input bit denoted by I_G and X_G. One can think of I_G as the garbled tables, which are formalized as garbled instruction set, whereas X_G refers to specific instructions, a subset of the instruction set that acts as the garbler input.

Step 3: Constructing multiple instances of the garbled MIPS core. The process continues by generating multiple, say, k, independent instances of the garbled MIPS core (i.e., garbled tables), each representing the same computational function, but garbled separately. The number of instances depends on the security level required by the designer [55] (see Section V-A2). Each instance contains the same logical operations but with different random values used for garbling,

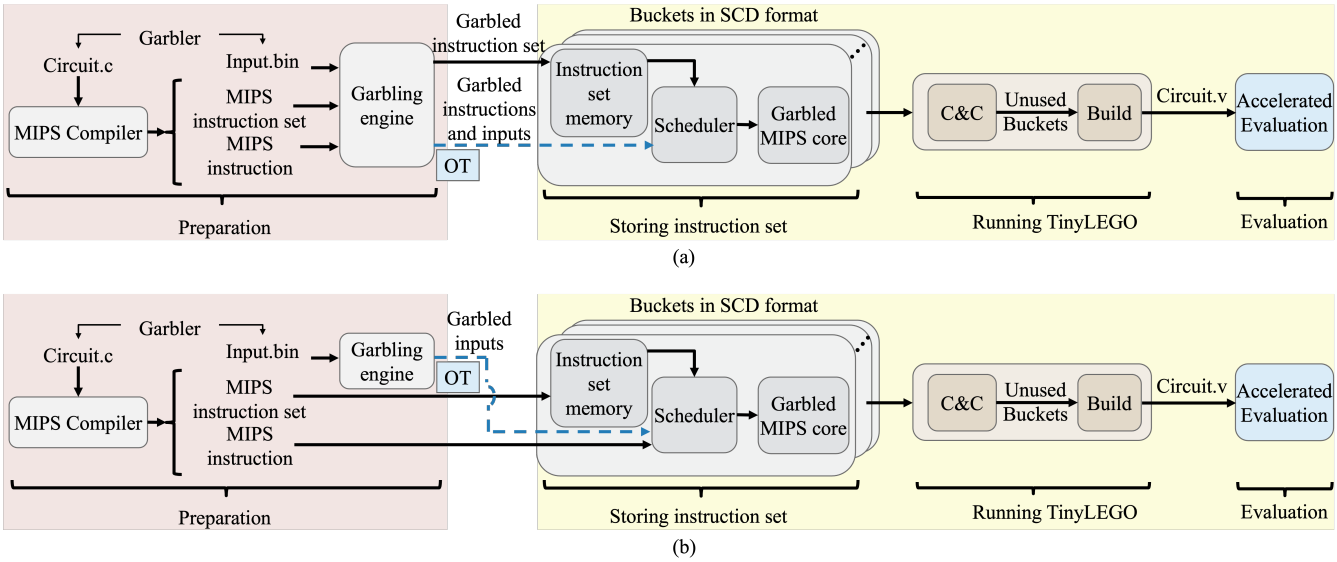


Fig. 6: A high-level flow of GuardianMPC. The processes highlighted in red and yellow run on the garbler's (NN publisher/provider) and user's machines. (a) Garbling the instructions and the instruction sets is included in the flow of oblivious inference. In the private training scenario, as shown in sub-figure (b), the instructions and the instruction sets are not garbled. The garbler's input (weights) are garbled and obliviously sent to the user via OT.

ensuring variability across instances. Later, the evaluator selects solely a subset of instances for verification (see Section V-A2).

The garbled instruction set and inputs are now ready to be stored in the next phase of the GuardianMPC flow, ensuring that the circuit function and data remain secure throughout the computation process.

Step 4: Memory organization for garbled instructions GuardianMPC leverages a structured memory layout that supports efficient access and retrieval while minimizing the interaction between the garbler and the evaluator. In this phase, garbled instruction set I_G and garbled inputs X_G prepared in preparation phase should be stored. The garbled instruction set I_G is placed in a designated memory unit called the `Instruction Set Memory`. This memory unit is designed to store encrypted instructions that allows for rapid access by the computation units within the GuardianMPC implementation. A significant advantage of storing the garbled instruction set within the `Instruction Set Memory` is the reduction in communication overhead. Traditional MPC protocols often require the garbler to interact with the evaluator during each phase of the computation, particularly when sending encrypted instructions or data. This back-and-forth interaction can be both time-consuming and vulnerable to various forms of communication-based attacks. By storing I_G and X_G locally within the GuardianMPC implementation, the system minimizes these interactions, although at the price of memory required on the user's device. Once the garbled instructions are securely stored, the `Scheduler` component of GuardianMPC can directly access the `Instruction Set Memory` to fetch the necessary instructions for processing. This reduces the latency and communication burden on the system without compromising the security since at least, the output should be decrypted by one interaction between the NN provider and the user.

2) Running TinyLEGO Protocol. : Following the steps below, GuardianMPC utilizes the LEGO protocol [45], [81] to ensure both input's consistency and correctness as well as the correctness of the garbled MIPS core's construction.

Step 1: Initializing the commitment scheme. For garbled instruction sets stored in the memory, the commitment scheme is initialized, which creates wire authenticators and commits to all associated wires. Here, the garbler commits to each wire of the garbled instruction sets.

Step 2: Checking correctness and integrity with cut-and-choose. The checking is performed through cut-and-choose running between the parties. To ensure the correctness of the garbled instances, the evaluator selects a random subset of garbled cores to be opened and verified. The selected instances are revealed entirely, allowing the evaluator to examine the internal structure and verify that the garbling process was conducted properly. Specifically, the evaluator checks the correctness of the encryption used for garbling each AND gate's truth table and ensures that the input mappings match the expected values. The garbler cannot predict which instances will be chosen for verification, thereby discouraging any attempt to manipulate the garbled core.

Furthermore, the wire authenticators are checked, where the corresponding gadget either accepts or rejects a given key (without revealing the value of the key) [81]. As demonstrated in [81], this additional step does not impose a cost in terms of time complexity thanks to the enhanced cut-and-choose pro-

cess that significantly reduces the overall cost (communication and time complexity). This guarantees that the inputs provided by the garbler are consistent with the committed values.

This step takes full advantage of function-independent preprocessing of [46], which relies on the fact that the preparation step can be taken in an offline fashion and done independently of the circuit. This matches perfectly the idea of implementing a general-purpose processor that is independent of the instructions corresponding to NN functionality to be evaluated in the online phase. The implementation of such a function-independent processor results in efficiency thanks to the parallelism in the preparation phase. Another advantage of that is adding more flexibility as the final output of offline processing is universal. Seen from the PFE perspective, the function-independent, general-purpose processor is a universal Turing machine [102] that is useful for evaluating a function privately. We should stress that so far, none of the LEGO protocols have been extended to support PFE; hence, GuardianMPC is the first to implement a LEGO protocol with PFE support.

Step 3: Building the final garbled core. After the verification step, if the opened instances pass the checks, the unopened instances are used to assemble the final GC. The number of unopened instances is proportional to the desired security parameter, which determines the statistical security of the protocol cf. [45]. More precisely, the cut-and-choose mechanism provides statistical assurance that for any fixed statistical security parameter s and any polynomial time-bounded adversary, the probability of an incorrect circuit going undetected is $2^{-s}+O(1)$. s is typically set to 40 [54], [81]. The final circuit is constructed by soldering together the remaining unopened instances, effectively combining them into a single correctly constructed MIPS core. This step ensures that each gate's output in one garbled instance is correctly wired to the input of the subsequent gates. At the end of this phase, the assembled GC is referred to as the protected garbled MIPS core. It encapsulates the logical functionality of the original MIPS architecture. The evaluator can now proceed to evaluate it with their inputs while remaining confident in the integrity of the GC.

3) Evaluation Acceleration: GuardianMPC employs hardware-based acceleration through a dedicated garbled MIPS evaluator [101], significantly improving the efficiency of the evaluation phase. The acceleration is crucial for minimizing the latency of evaluating GCs, especially in large computations like deep learning inference.

The garbled MIPS core is designed with various hardware primitives to optimize the evaluation of encrypted instructions. It utilizes `Arithmetic Logic Units (ALUs)` specifically configured for performing operations on encrypted values, ensuring that computations remain secure throughout the process. `Lookup Tables (LUTs)` are employed for the efficient decoding of garbled values, which allows for quick access to encrypted data during evaluation. Additionally, the design incorporates `on-chip memory` to store intermediate results and garbled labels, reducing the reliance on external memory – a common performance bottleneck – and thereby further accelerating the overall evaluation process.

The evaluation phase involves both the garbler and evaluator engaging in a secure exchange of encrypted input labels using the OT protocol; see Figure 6. Once the evaluator obtains the garbled labels for their inputs, they proceed to evaluate the GC by computing the encrypted outputs of each gate using the garbled tables (see Figure 1. Throughout this phase, the garbled MIPS evaluator ensures that neither the user nor the NN provider gains access to the underlying secrets of one another. The evaluator ultimately decrypts the garbled outputs using the decryption keys provided by the garbler, allowing them to retrieve the final results without exposing any intermediate computation values. In this respect, this combination of hardware acceleration and secure computation protocols makes GuardianMPC a robust solution.

B. GuardianMPC's Functionalities

GuardianMPC offers two primary functionalities: oblivious inference and private training. Both functionalities utilize the GuardianMPC flow, but differ in inputs, some processes, and goals. This section provides a detailed explanation of each functionality and its implementation.

Oblivious inference. During the evaluation, the user provides their input data, which is encrypted into garbled labels and obliviously transferred to the evaluator using OT. The evaluation of the garbled model is then performed on a hardware platform, e.g., an FPGA. The time complexity of oblivious inference with GuardianMPC is polynomial in the size of the garbled instruction set and size of inputs (I_G and X_G, respectively), the security parameter, and the degree of parallelism provided by the hardware-accelerated MIPS core on the FPGA cf. [103], [104]. The parallelization factor allows for concurrent processing of garbled instructions, leading to a faster evaluation phase.

Private training. Private training (see Section I) focuses on training an NN with privacy, i.e., while keeping both the training data and the model weights private. The training process involves multiple iterations (epochs), where the model is updated based on the training data. Unlike oblivious inference, the input provided to GuardianMPC for training initially consists of randomized values, which serve as placeholders for the garbled training dataset. The circuit function `circuit.c` is extended to include both forward and backward propagation computations, enabling the model to learn from data over successive epochs.

The entire training process is performed on the server side, where the evaluator (representing the data holder) sends the garbled training dataset to the server via OT, ensuring that only encrypted labels are exchanged. The garbler, representing the model owner, receives these labels and uses the garbled MIPS core to compute the forward pass, followed by the backward pass for each batch of data. During each epoch, the server updates the model weights by adjusting them according to the gradients derived from the backward pass. The updated weights remain encrypted throughout this process, preventing the server from accessing the underlying values. At the end

of each epoch, the server sends the updated garbled weights back to the evaluator as the encrypted output. The iterative nature of training makes this process more computationally intensive than inference, as each epoch requires both forward and backward computations along with weight updates. The time complexity of private training using GuardianMPC is polynomial in size of garbled inputs, X_G, the security parameter, and the degree of parallelism offered by the hardware-accelerated MIPS core on the FPGA cf. [103], [104].

C. Differences between GuardianMPC and TinyLEGO

GuardianMPC and TinyLEGO share many common features: both exhibit modularity, easy adjustments to various applications, optimized low latency and high throughput. Both TinyLEGO [81] and GuardianMPC are based on the LEGO framework [45], although key differences exist as follows.

Architecture and its privacy. GuardianMPC and TinyLEGO [81] are both built on the foundational principles of the LEGO protocol [45], focusing on secure 2PC using GCs. However, they differ significantly in their approach. GuardianMPC offers PFE. GuardianMPC focuses on converting functions into MIPS instructions, which are then garbled and evaluated securely and privately. This low-level approach ensures that the function logic remains hidden during the computation process. It allows GuardianMPC to obfuscate the underlying computational processes, providing an extra layer of privacy and preventing an adversary from understanding the details of the function being computed.

On the other hand, TinyLEGO [46], [81] adopts a higher abstraction level in constructing GCs, allowing it to focus on the gate level, where each logical operation (such as NAND, XOR, etc.) is garbled directly. This approach simplifies the garbling process since it does not require translating the entire function into a lower-level representation like MIPS instructions. Yet, TinyLEGO cannot support PFE as it is intended to only offer secure function evaluation. Although TinyLEGO provides strong security against the manipulation of weights via its robust cut-and-choose mechanism, it does not inherently hide the function logic, i.e., network architecture. Consequently, for oblivious inference, TinyLEGO cannot prevent the malicious adversary from inserting architectural backdoors as opposed to GuardianMPC.

Applicability and overhead. GuardianMPC introduces an additional computational layer by translating complex functions into MIPS instructions before garbling them. This conversion process can introduce overhead, as it requires additional steps to transform high-level functions into a lower-level representation. The benefit of this approach lies in its robustness against function exposure, but it comes at the cost of increased preprocessing time and resource requirements. TinyLEGO, with its direct approach to garbling circuits at the gate level, is designed for efficient garbling [46], [81]. By focusing on reducing communication rounds to the optimal two rounds and leveraging efficient cut-and-choose mechanisms, TinyLEGO can perform garbling with relatively low overhead.

Flexibility. The flexibility of GuardianMPC is centered on its use of MIPS instructions, allowing it to handle a diverse range of functions by adapting them into a universal instruction format. This makes GuardianMPC particularly suited for scenarios where function privacy is crucial, as it can adapt to various proprietary algorithms or computations that need to remain hidden during secure evaluation. However, this reliance on MIPS instruction sets may limit its adaptability to non-MIPS architectures or cases where direct circuit representation is desired. TinyLEGO [46], [81] is designed with flexibility in terms of integrating with a wide range of secure computation workflows. This adaptability is particularly useful when the priority is to achieve fast computation and secure interaction without the need for low-level function obfuscation (see Section VII-C for further discussion).

VI. Experimental Results

GuardianMPC is coded in C++ and is available on GitHub [105]. We evaluate the performance of GuardianMPC in two scenarios: oblivious inference and private training. For the former, we assess the performance of GuardianMPC to evaluate NN models such as multi-layer perceptrons (MLPs) and convolutional NNs (CNNs) on garbled data while ensuring both input privacy and function hiding. The oblivious inference and private training results are obtained by considering images in the MNIST dataset [106]. For private training, we employ GuardianMPC to securely train an NN on private data, ensuring that no information about the training data or model parameters is disclosed during the training process.

A. Experimental Setup

In both oblivious inference and private training scenarios, the garbler used a machine equipped with Intel Xeon Silver 16 core CPU @2.5GHz, NVIDIA RTX-A4000 GPU, 128 GB RAM, and Linux Ubuntu 20. As for private training, the evaluator was an Intel Core i7-7700 CPU @ 3.60GHz system with 16 GB RAM. The garbler runs the compiled version of TinyLEGO [107] framework, which is written in C++. The evaluator running the oblivious inference was an Artix-7 FPGA device (XC7AT100T) using Xilinx Vivado Design Suite 2021 [108]. The garbler and evaluator were connected via a local area network (LAN) in the same region. The average network delay is 0.2 ms and the bandwidth is 1 GB/s, a similar network utilized by [18].

B. Oblivious Inference

The results presented in this section are for oblivious inference, where the aim is to evaluate NNs (MLPs and CNNs) on garbled data, ensuring both input and function privacy; consequently, the attacker is hindered from inserting backdoors in pre-trained models (see Figure 5).

1) Benchmark Models: **MLPs.** For oblivious inference, we considered two MLP models. The first one, hereafter called BM1, is an MLP with 784 neurons in the input layer, three hidden layers of 1024 neurons each, and an output layer with 10 neurons trained on MNIST [106]. The second one, called

TABLE III: Comparison between the execution time of BM1 (the numbers in boldface indicate the best results).

Approach	Security	# of AND gates	Preparation [ms]				Online [ms]				
			Construction	BaseOT	Random Generation	Total	Communication	Checking	Building	Evaluation	Total
Baseline	None	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	**3.43**
TinyLEGO [107]	SFE	2098	**3591.8**	**579.03**	**17.59**	**4188.42**	4219.72	**492.7**	**6.09**	43.84	4762.36
GuardianMPC	PFE	2098	5912.29	719.88	18.12	6650.29	**801.19**	608.94	6.23	**3.26**	1419.62

TABLE IV: Comparison between the execution time of BM2 (the numbers in boldface indicate the best results).

Approach	Security	Preparation [ms]	Online [ms]			Total [ms]
			Comm.	Evaluation	Total	
Baseline	None	N/A	N/A	N/A	**171.39**	**171.39**
MiniONN [20]	SFE	**880**	N/R	N/R	400	1280
TinyLEGO [107]	SFE	1961.24	70.57	34.76	392.39	2353.63
GuardianMPC	PFE	2323.96	**10.23**	**2.8**	425.91	2749.87

TABLE V: Comparison between the execution time of BM3 (the numbers in boldface indicate the best results).

Approach	Security	Preparation [s]	Online [s]			Total [s]
			Comm.	Evaluation	Total	
Baseline	None	N/A	N/A	N/A	**9.72**	**9.72**
MiniONN [20]	SFE	472	N/R	N/R	72	544
EzPC [110]	SFE	N/R	N/R	N/R	N/R	265.6
TinyLEGO [107]	SFE	**913**	40.22	7.29	154	1067
GuardianMPC	PFE	1191	**23.73**	**1.18**	208	1399

TABLE VI: Comparison between the execution time of LeNET-5 [111] (the numbers in boldface indicate the best results).

Approach	Security	Preparation [s]	Online [s]			Total [s]
			Comm.	Evaluation	Total	
Baseline	None	N/A	N/A	N/A	**1.73**	**1.73**
TinyGarble2 [97]	SFE	N/R	N/R	N/R	91.1	91.1
TinyLEGO [107]	SFE	**387.85**	37.02	2.61	49.23	419.95
GuardianMPC	PFE	429.23	**6.91**	**0.4**	36.95	466.25

BM2, is an MLP used as a benchmark NN in [18], consisting of a single hidden layer with 128 neurons and square activation function, trained on MNIST [20].

CNNs. We also implemented CNNs to test the scalability of the frameworks for oblivious inference. The first one, BM3, is a seven-layer CNN used by Chameleon [109], EzPC [110], and MiniONN [20], trained on CIFAR-10. The second one, LeNET-5, is a CNN used in TinyGarble2 [97] for MNIST classification [111]. LeNET-5 consists of convolutional layers followed by fully connected layers.

The benchmark NNs were implemented in three different ways:

- **Baseline.** Benchmarks were implemented without any security measures (without garbling scheme or function hiding).
- **Within the TinyLEGO framework (SFE) [107].** Benchmarks were implemented without function hiding.
- **Protected, with function hiding (PFE).** Benchmarks were implemented using GuardianMPC, which ensures function hiding and protection against malicious adversaries. This resembles the oblivious inference scenario, where inserting backdoors in pre-trained models is prevented.

The following subsections discuss the performance of GuardianMPC in terms of execution time for both the preparation and online phases. The results are compared with the most relevant studies, which have employed the same benchmarking NNs.

2) Execution Time: **BM1.** Table III compares the time complexity for BM1 embedded in different frameworks. The online phase, consisting of checking, building, and evaluation, shows that TinyLEGO incurs $149\times$ more execution time than the unprotected FPGA-based accelerator. GuardianMPC requires more preparation and online time than TinyLEGO. This additional cost is due to the enhanced privacy provided by GuardianMPC, including function hiding, which is crucial for oblivious inference.

BM2. Table IV presents the results for BM2. GuardianMPC requires 6% more online execution time than MiniONN, which benefits from single instruction multiple data (SIMD) optimization. However, GuardianMPC offers stronger privacy against malicious adversaries, whereas MiniONN is only resilient to semi-honest adversaries. These results highlight the trade-off between performance and privacy guarantees in oblivious inference settings.

BM3. Table V shows the time spent running the frameworks against BM3. GuardianMPC shows a higher online execution time compared to TinyLEGO and MiniONN, but provides the strongest privacy guarantee. The online time for GuardianMPC is $21.3\times$ higher than the FPGA-based unprotected implementation. However, it improves privacy by hiding the function and protecting both the input and the intermediate data in the CNN inference task.

LeNET-5. Table VI compares LeNET-5 implemented within different frameworks. GuardianMPC incurs $21.4\times$ more online execution time than the unprotected FPGA-based implementation. Nevertheless, it significantly outperforms TinyGarble2 [97], being $1.46\times$ faster during the online phase. This is thanks to the parallelization supported by the hardware platform (FPGA), which increases the efficiency compared to a software implementation (TinyGarble2). Moreover, the advantage of GuardianMPC lies in its ability to handle oblivious inference tasks while maintaining high levels of protection and acceptable performance trade-offs.

Evaluation acceleration on FPGAs. One of the key contributions of GuardianMPC is its ability to accelerate the evaluation phase of the LEGO protocol [45], which is crucial for oblivious inference. The hardware accelerator takes advantage of hardware optimization techniques and the FPGA's parallel processing capabilities. Compared to TinyLEGO, according to Table III and Table IV, GuardianMPC accelerates the evaluation phase by $13.44\times$ and $12.41\times$ for BM1 and BM2, respectively. Similarly, GuardianMPC accelerates the

TABLE VII: Comparison between the time complexity in training phases of BM4 using GuardianMPC and SecureML [18] for 15 epochs using ReLU activation function (the numbers in boldface indicate the best results).

Approach	Security	Preparation [s]	Online [s]			Total [s]
			Comm.	Evaluation	Total	
SecureML [18]	SFE	**290000**	N/R	N/R	4239.7	**294239.7**
GuardianMPC	SFE	367240	833	42	**875**	368115

evaluation phase for CNN benchmarks such as BM3 and LeNET-5 by $6.17\times$ and $6.52\times$, respectively (see Tables V-VI). This demonstrates the potential of GuardianMPC to handle oblivious inference efficiently, especially in complex NN architectures like CNNs.

C. Private Training

1) Benchmark Model: We trained a fully-connected NN on the MNIST dataset using our privacy-preserving GuardianMPC protocol. The architecture of the NN includes two hidden layers, each containing 128 neurons, with ReLU and square functions as the activation functions. In the output layer, we employed a softmax function as in SecureML [18], and the cost function used for training was the cross-entropy loss function. The labels were encoded as one-hot vectors with 10 elements corresponding to the 10 classes of the MNIST dataset, hereafter called BM4. For the sake of comparison, we replicated the exact NN and batch size configuration in SecureML [18]. The batch size $|B|$ was set to 128, and the training process was conducted for 15 epochs, similar to the setup described in SecureML [18].

2) Execution Time: Table VII presents the execution time of training phases when running BM4 in GuardianMPC and SecureML [18] frameworks. A key observation from this table is that GuardianMPC significantly reduced the online time to 875s compared to SecureML's online time, which is 4239.7s. This improvement shows that GuardianMPC's online phase takes nearly $\times 4.8$ faster than SecureML, attributed to two primary factors: batch-wise data transmission over the local LAN and the inherent advantages of LAN-based communication, as explained below.

Batch-wise data transmission over LAN. GuardianMPC benefits from transmitting data in mini-batches, with a batch size $|B| = 128$. This approach ensures that the model processes smaller, manageable data segments incrementally rather than waiting for the entire dataset to be processed simultaneously. The mini-batch strategy enhances computational efficiency by enabling parallelism, where operations for different batches are computed concurrently. In contrast, if data were processed in larger chunks or sequentially, it would increase communication latency, thereby extending the online phase duration.

Since GuardianMPC processes batch one at a time, it minimizes the data transmitted in each communication round, ensuring a faster exchange between the garbler and evaluator. This incremental data flow keeps the communication pipeline active throughout the computation, reducing idle times and maximizing resource utilization. Consequently, GuardianMPC achieves lower communication overhead, contributing to a faster online phase compared to SecureML [18].

Impact of LAN. The experiments were conducted considering a LAN, which provides a low-latency and high-bandwidth communication channel between the garbler and evaluator. This setup offers a significant advantage over wide-area networks (WANs) or cloud-based setups, where higher latency and bandwidth limitations could adversely affect communication performance. LAN-based communication ensures minimal delay during the exchange of garbled inputs and encrypted outputs, directly benefiting the online phase of GuardianMPC.

The reduced online time in GuardianMPC can be partly attributed to how we configured a low-latency, efficient communication over LAN. With each batch of data sent locally, the need for long waiting times between consecutive communication rounds is eliminated, leading to seamless, uninterrupted computation. Despite running the protocol via LAN, SecureML [18] may not have fully exploited this LAN advantage. GuardianMPC's focus on batch-wise data transfer within a LAN environment ensures a faster online phase, as seen in the comparison.

Preparation overhead. The reduced online time in GuardianMPC comes at the cost of a $\times 1.26$ increased preparation time. This overhead is due to the GuardianMPC's design, which introduces additional steps to translate high-level NN operations into MIPS instructions. Moreover, GuardianMPC extends the cut-and-choose mechanism of TinyLEGO [107] to enhance security by preparing multiple cores and verifying a subset of them for integrity. This can also contribute to the computational overhead during preparation. This step ensures that computations in the online phase are efficient and secure against a malicious adversary that attempts to insert a backdoor in the NN model.

The preparation phase also involves batch-wise optimizations to align data and circuits for efficient execution. While improving online performance, these optimizations contribute further to the preparation overhead due to the setup required for batch processing. Despite the longer preparation time, this design shifts the computational burden to the offline phase, ensuring that the online phase is smooth and uninterrupted.

The result of this trade-off is evident in the performance comparison: GuardianMPC's online phase is $\times 4.8$ faster than SecureML [18]. In contrast, the total computation time shows only a $\times 1.25$ increase due to the preparation overhead. GuardianMPC's approach demonstrates the benefits of transferring overhead to the preparation stage to enable faster real-time performance during the online phase. Additionally, as explained next, GuardianMPC does not affect the predictive performance of the NN during the private training.

3) Accuracy Evaluation: To assess the predictive performance of the BM4 model, we trained it for 15 epochs using both GuardianMPC and PyTorch [112], i.e., the plaintext training as adopted in SecureML [18]. After completing the training phase in GuardianMPC, we extracted the resulting

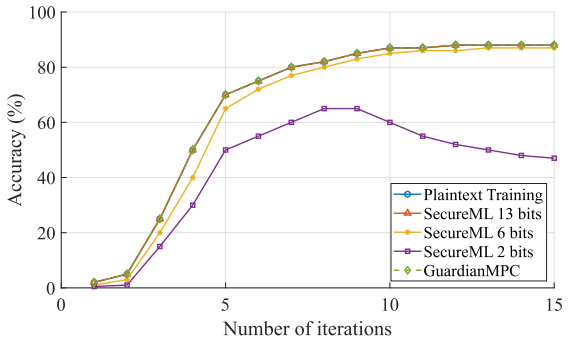


Fig. 7: Comparison of accuracy over the first 15 iterations between plaintext training, SecureML [18] at various bit precisions (13, 6, and 2 bits), and GuardianMPC trained on MNIST [106] dataset.

NN and applied it against the MNIST test set [106], also used for the PyTorch evaluation. The aim was to validate whether the private training process in GuardianMPC introduces any degradation in accuracy. Both models shared the same architecture, activation functions, and parameters with SecureML [18] to ensure consistency. The GuardianMPC-generated NN was loaded into a PyTorch model to facilitate comparison, ensuring no discrepancies arose from framework differences.

In SecureML [18], different levels of bit precision were used to balance computational efficiency and accuracy. Training with 13 bits closely matches plaintext training, while lower precision, such as 6 bits and 2 bits, introduces more accuracy degradation. However, as evident from Figure 7, GuardianMPC maintains identical accuracy to plaintext training, demonstrating that the secure computation process does not compromise the model's predictive performance, unlike lower bit-precision methods used in SecureML [18]. This is due to the fact that SecureML has a minor accuracy loss due to the effect of truncation, polynomial approximations, and modified activation functions employed in the NNs. On the contrary, GuardianMPC uses a floating point for data without any truncation. Moreover, GuardianMPC does not apply approximation as it converts the high-level description to GC using MIPC instructions. Finally, GuardianMPC does not require a modified activation function.

VII. Discussion on GuardianMPC's Features

A. Training Accuracy of GuardianMPC

GuardianMPC ensures that the accuracy of NNs remains unaffected during outsourced training by avoiding techniques known to degrade accuracy. Unlike other privacy-preserving methods, such as Homomorphic Encryption (HE) [113], GuardianMPC eliminates the need for truncation [114], polynomial approximations [18], and modified activation functions [22], all of which can introduce errors. Truncation, often used to simplify encrypted arithmetic, leads to cumulative errors during training, affecting model convergence and accuracy [18], [32]. Similarly, frameworks relying on polynomial approximations of non-linear activation functions, such as ReLU or sigmoid, experience degraded precision, especially with low-degree polynomials [32]. GuardianMPC bypasses these issues by directly garbling standard functions, ensuring exact computation without approximation. Furthermore, GuardianMPC avoids replacing activation functions with simplified alternatives, which can alter the model's dynamics and reduce accuracy [17], [115]. By computing with the original activation functions, such as ReLU and softmax, GuardianMPC guarantees the secure model's behavior follows the plaintext model, maintaining predictive performance throughout the training process.

We stress that the accuracy of NN models is inherently influenced by the quality of the input data. For instance, low-quality inputs, such as blurry or low-resolution images, are known to degrade model accuracy due to the loss of critical information during pre-processing or imaging. While the accuracy of the NN may be reduced when presented with such low-quality inputs, this reduction is solely a result of the model's limitations in handling data and not a consequence of the secure computation process performed by GuardianMPC. GuardianMPC, however, operates as a secure computation framework that does not alter the underlying functionality or behavior of the NN model. Specifically, GuardianMPC processes inputs in their original form using secure multiparty computation techniques, preserving the data privacy and the integrity of the model.

B. GuardianMPC vs. Other Countermeasures

HE vs. GuardianMPC. HE [113] emerges as a promising solution for NN, allowing computations on encrypted data. Despite the potential, HE-based NN accelerators face challenges such as the complex implementation of non-linear functions like ReLU, significant computational complexity, and truncation errors [18], [29]. As stated in [24], both HE and GC have inherent limitations. The major limitation of HE is its high computational complexity, which grows with the depth of the arithmetic circuits required. This complexity leads to significant performance issues, making it less feasible for real-time applications. On the other hand, GCs are computationally more efficient as they use symmetric-key cryptographic primitives and benefit from hardware support. However, the primary drawback of LEGO [45], the core of GuardianMPC, is that it bears communication overhead, as it requires the exchange of large amounts of data between parties. This aspect makes it less suitable for scenarios where bandwidth is limited. While conventional HE methods may lack the circuit privacy of Private Function Evaluation (PFE) offered by GuardianMPC, fully HE protocols achieve this goal [116], [117].

Zero-knowledge proof vs. GuardianMPC. The limitation of ZK proofs lies predominantly in their practical efficiency [73], [74]. While theoretically robust in offering security guarantees, ZK Proofs tend to be computationally intensive and resource-demanding [74]. This is particularly evident in scenarios

requiring security against malicious adversaries. Their complexity makes them less practical for applications that demand efficient and real-time computations. The GuardianMPC uses LEGO protocol that addresses these limitations by offering an efficient mechanism for constructing secure two-party computations, making it more suited for applications involving large circuits and the need for robust protection against malicious adversaries [81].

Trigger reconstruction vs. GuardianMPC. NeuralCleanse [118] developed the first trigger reconstruction approach [119]. Trigger reconstruction in machine learning aims to detect and remove backdoor triggers by modifying inputs to find misclassification patterns, indicating potential backdoors [118]. While this can identify and help remove these triggers, the process has limitations. Detecting complex or well-disguised triggers that blend into the NN model is challenging and computationally intensive. Additionally, there is no guarantee of completely removing the backdoor, especially in models with complex features, making it unreliable against all backdoor attacks [119].

On the other hand, GuardianMPC, through its implementation of the LEGO protocol [45], offers a robust solution to stop the adversary from inserting backdoors instead of detecting them. As precisely formulated in [90], detection of backdoors is strictly less desirable than protecting the NN in the first place. Unlike trigger reconstruction, GuardianMPC efficiently manages computational load by offloading intensive computation to the pre-processing phase. This strategic distribution of computational load ensures that its online phase delivers near real-time performance, making it well-suited for NN interfaces where quick response times are crucial.

Model inspection vs. GuardianMPC. Model inspection in machine learning is a process for detecting backdoor attacks by analyzing a trained model's neurons, outputs, and overall behavior. It involves examining neuron activations for anomalies, assessing the model's responses to various inputs, and utilizing advanced detection techniques, including NN-based anomaly detection and interpretability tools [119]. NeuronInspect [120], DeepInspect [121], and Meta Neural Trojan Detection (MNTD) framework [122] are the cutting-edge approaches that are built based on the model inspection [119]. Despite the strength of model inspection in identifying backdoor insertion attacks in machine learning models, it faces several limitations. Primarily, it is most effective for certain types of NN models, like CNNs, and might not perform as well for other network architectures or data modalities [119]. It also assumes fixed backdoor patterns, making it less effective against more sophisticated, adaptable triggers. Another key challenge is its high computational complexity, particularly evident in methods like the MNTD framework [122], which involves training numerous shadow models. These limitations impose difficulties in applying model inspection universally, especially against advanced and evolving backdoors that may not adhere to predictable patterns [119].

In contrast to model inspection, GuardianMPC presents a more universally effective solution against backdoor insertion attacks across all types of NNs. This is possible thanks to the variant of the LEGO protocol [46] that garbles the circuits independent from their functionality and model of the NN. Additionally, GuardianMPC's strategy of managing computational load between its online and offline phases offers a significant advantage over frameworks like MNTD [122]. This efficient management of computational resources enhances performance and makes GuardianMPC more practical for deployment in real-world scenarios.

Logic obfuscation vs. GuardianMPC. In the context of securing NN accelerators, logic obfuscation [123], [124] is presented in [125] as a countermeasure against hardware Trojans. This method involves adding redundant components or a locking mechanism to ensure it continues functioning even if a Trojan compromises some components or is completely rendered useless without the correct key [125]. However, while effective in certain scenarios, ad hoc logic obfuscation has its limitations, notably in its potential for increasing complexity and cost in the design and verification processes and possibly reducing the overall performance due to additional logic gates or increased power consumption [125].

Unlike logic obfuscation, GuardianMPC offers a theoretically-sound and robust alternative. It focuses on using MPC techniques, specifically GC, to secure the function's high-level logic effectively. This not only secures the underlying operations of NN models [101] but also ensures that even if an insider has access to the design, they cannot easily understand or tamper with the logic. Furthermore, GuardianMPC's approach is inherently suited to protecting against sophisticated backdoor attacks by garbling the function's core operations, providing a dual layer of security–both in function security and privacy, as well as hardware security.

Formal verification vs. GuardianMPC. Formal verification [126], as discussed in [125], serves as a crucial defense against hardware Trojans by rigorously using mathematical techniques to verify the integrity of hardware designs. However, its effectiveness may be limited when applied to complex ML hardware accelerators with specialized hardware and novel architectures, which present unique challenges that formal methods still need to address [125]. On the contrary, GuardianMPC leverages MPC to provide a more flexible and robust security layer. This approach does not rely solely on the initial design correctness, but continuously protects the data and computation processes.

1) Qualitative comparison: We have analyzed the above-mentioned mitigation scenarios with regard to maintaining the accuracy level, computational load, scalability, and universality to highlight the advantages of using GuardianMPC. Table VIII illustrates a qualitative comparison between GuardianMPC and cutting-edge mitigation against direct weight manipulation and architectural backdoor insertion. As observable in Table VIII, only GuardianMPC maintains its real-time performance. Moreover, GuardianMPC offers a scalable solution independent of the NN model. This makes GuardianMPC a better fit for a backdoor-resilient NN accelerator than other

TABLE VIII: Comparative analysis of various security benchmarks in data processing.

Approach	Maintain Accuracy	Scalability	Real-time Performance	Model Independency
Homomorphic Encryption [17], [18], [20], [24]	redN	greenY	greenY	greenY
Zero-knowledge proof [54], [70]	greenY	redN	redN	greenY
Byzantine-resilient [127]	redN	greenY	greenY	greenY
Trigger reconstruction [118]	greenY	redN	redN	greenY
Model inspection [120]–[122]	greenY	redN	redN	redN
Logic obfuscation [123], [124]	greenY	redN	greenY	greenY
Formal verification [126]	greenY	redN	redN	greenY
GuardianMPC	greenY	greenY	greenY	greenY

mitigation approaches.

C. GuardianMPC's Implementation Flexibility and Scalability

While GuardianMPC leverages FPGA-based implementations to achieve hardware acceleration and efficient computation, its design is not strictly limited to FPGAs. The framework utilizes an encrypted version of instructions, which allows for the potential adaptation to other processor architectures that support custom instruction decoding. Specifically, if a processor architecture, such as ARM [128], allows users to modify the decode phase in order to accept encrypted instructions, GuardianMPC can be seamlessly implemented on such processors. In this scenario, the FPGA used in the current implementation could be replaced with a compatible processor that incorporates this functionality. This approach would retain the core privacy-preserving and secure computation principles of GuardianMPC while broadening its accessibility to users who do not have access to specialized hardware like FPGAs. Additionally, this flexibility in hardware selection ensures that GuardianMPC remains a scalable and adaptable framework capable of meeting the requirements of diverse computational environments. Future research can explore hybrid approaches that integrate software-based solutions with minimal hardware modifications to support encrypted instruction decoding. This would further enhance the accessibility and usability of GuardianMPC across a wider range of devices.

Moreover, the scalability of GuardianMPC to large and complex NN architectures, such as transformers, is an important consideration for extending its use to a broader range of applications. While the experimental results in this work focus on usual benchmarking models like those trained on the MNIST datasets [106], GuardianMPC's design inherently supports scalability to larger architectures, with some trade-offs. In the case of private training, the training process is conducted entirely on the server side, which involves iterative computations across multiple epochs. For more complex NNs, the number of model parameters and hyperparameters can proportionally increase the computational cost of the offline phase. This may extend the preparation time as the GC's size and associated cryptographic operations scale up with the complexity of the network.

For oblivious inference, larger NNs with deeper architectures and more parameters result in higher communication costs due to the increased size of the garbled instruction sets and inputs. This, in turn, impacts the online computation time, as the evaluator must process a larger volume of encrypted data. However, advanced communication protocols can significantly alleviate this bottleneck by optimizing the data transfer process, such as by employing batch-wise transmission [18], parallel communication streams [109], or hardware-accelerated protocols [24], [31], [41]. These optimizations ensure that the increase in online computation time remains manageable, thereby maintaining the scalability of GuardianMPC for larger NN architectures. By leveraging these advanced communication and optimization techniques, GuardianMPC can effectively scale up to accommodate more complex NNs, such as transformers, without compromising the privacy and security guarantees of the NN. Future work can explore the implementation of these techniques to further validate the framework's scalability.

VIII. Conclusion

In this work, we introduced GuardianMPC, a backdoor-resilient NN accelerator that offers secure and private 2PC for oblivious inference and private training. While leveraging the LEGO protocol to defend against backdoor insertion threats by malicious adversaries, GuardianMPC's unique implementation of private function evaluation makes it a promising solution to combat the effect of direct weight manipulation and architectural backdoors. By integrating advanced GC optimizations and harnessing FPGA computational power, GuardianMPC fortifies NN models with robust security and high efficiency. GuardianMPC achieves up to $13.44\times$ faster evaluation phase performance than TinyLEGO, a LEGO-based framework, with only minimal extra offline computation. This combination of advanced cryptography and high-performance hardware makes GuardianMPC a secure, resilient, and efficient solution for NN computation in today's evolving security landscape.

Acknowledgment

This work has been partially supported by Semiconductor Research Corporation (SRC) under Task IDs 2991.001 and 2992.001 and NSF under award number 2138420.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] Amazon, "Machine Learning on AWS." [Online] https://aws.amazon.com/machine-learning/ [Accessed: Oct.16, 2024], 2023.

[3] Microsoft Corp., "Azure Machine Learning." [Online] https://azure.microsoft.com/en-us/products/machine-learning/ [Accessed: Oct.16, 2024], 2023.

[4] "Caffe Model Zoo." [Online] https://github.com/BVLC/caffe/wiki/Model-Zoo [Accessed: Oct.16, 2024], 2012.

[5] "Keras Pre-trained Models." [Online] https://keras.io/applications/ [Accessed: Oct.16, 2024], 2012.

[6] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25th Annual Network And Distributed System Security Symposium (NDSS 2018)*, Internet Soc, 2018.

[7] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent backdoor attacks on deep neural networks," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 2041–2055, 2019.

[8] Y. Gao, B. G. Doan, Z. Zhang, S. Ma, J. Zhang, A. Fu, S. Nepal, and H. Kim, "Backdoor attacks and countermeasures on deep learning: A comprehensive review," *arXiv preprint arXiv:2007.10760*, 2020.

[9] R. Tang, M. Du, N. Liu, F. Yang, and X. Hu, "An embarrassingly simple approach for trojan attack in deep neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 218–228, 2020.

[10] X. Gong, Z. Wang, Y. Chen, M. Xue, Q. Wang, and C. Shen, "Kaleidoscope: Physical backdoor attacks against deep neural networks with rgb filters," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[11] M. Bober-Irizar, I. Shumailov, Y. Zhao, R. Mullins, and N. Papernot, "Architectural backdoors in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24595–24604, 2023.

[12] T. Clifford, I. Shumailov, Y. Zhao, R. Anderson, and R. Mullins, "Impnet: Imperceptible and blackbox-undetectable backdoors in compiled neural networks," *arXiv preprint arXiv:2210.00108*, 2022.

[13] Y. Li, J. Hua, H. Wang, C. Chen, and Y. Liu, "Deeppayload: Black-box backdoor attack on deep learning models through neural payload injection," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 263–274, IEEE, 2021.

[14] A. Warnecke, J. Speith, J.-N. Möller, K. Rieck, and C. Paar, "Evil from within: Machine learning backdoors through hardware trojans," *arXiv preprint arXiv:2304.08411*, 2023.

[15] T. A. Odetola, H. R. Mohammed, and S. R. Hasan, "A stealthy hardware trojan exploiting the architectural vulnerability of deep learning architectures: Input interception attack (iia)," *arXiv preprint arXiv:1911.00783*, 2019.

[16] J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," *arXiv preprint arXiv:1806.05768*, 2018.

[17] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Intrl. Conf. on machine learning*, pp. 201–210, PMLR, 2016.

[18] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE symposium on security and privacy (SP)*, pp. 19–38, IEEE, 2017.

[19] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[20] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. of the 2017 ACM SIGSAC Conf. on computer and Comm. security*, pp. 619–631, 2017.

[21] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *International Conference on Machine Learning*, pp. 812–821, PMLR, 2019.

[22] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *arXiv preprint arXiv:1811.09953*, 2018.

[23] A. Sanyal, M. Kusner, A. Gascon, and V. Kanade, "Tapas: Tricks to accelerate (encrypted) prediction as a service," in *International conference on machine learning*, pp. 4490–4499, PMLR, 2018.

[24] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th USENIX Security Symp. (USENIX Security 18)*, pp. 1651–1669, 2018.

[25] Q. Lou and L. Jiang, "She: A fast and accurate deep neural network for encrypted data," *Advances in neural information processing systems*, vol. 32, 2019.

[26] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III 38*, pp. 483–512, Springer, 2018.

[27] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proc. of the 55th annual design automation Conf.*, pp. 1–6, 2018.

[28] M. Ball, B. Carmer, T. Malkin, M. Rosulek, and N. Schimanski, "Garbled neural networks are practical," *Cryptology ePrint Archive*, 2019.

[29] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "{XONN}:{XNOR-based} oblivious deep neural network inference," in *28th USENIX Security Symp. (USENIX Security 19)*, pp. 1501–1518, 2019.

[30] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "Chet: an optimizing compiler for fully-homomorphic neural-network inferencing," in *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, pp. 142–156, 2019.

[31] W. Z. Srinivasan, P. Akshayaram, and P. R. Ada, "Delphi: A cryptographic inference service for neural networks," in *Proc. 29th USENIX Secur. Symp*, pp. 2505–2522, 2019.

[32] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 325–342, 2020.

[33] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symp. on Foundations of Computer Science (sfcs 1986)*, pp. 162–167, IEEE, 1986.

[34] L. K. Ng and S. S. Chow, "Sok: Cryptographic neural-network computation," in *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 497–514, IEEE, 2023.

[35] Z. Á. Mann, C. Weinert, D. Chabal, and J. W. Bos, "Towards practical secure neural network inference: the journey so far and the road ahead," *ACM Computing Surveys*, vol. 56, no. 5, pp. 1–37, 2023.

[36] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[37] N. Agrawal, A. Shahin Shamsabadi, M. J. Kusner, and A. Gascón, "Quotient: two-party secure neural network training and prediction," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1231–1247, 2019.

[38] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "{ABY2. 0}: Improved {Mixed-Protocol} secure {Two-Party} computation," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2165–2182, 2021.

[39] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 35–52, 2018.

[40] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4961–4973, 2021.

[41] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *arXiv preprint arXiv:2004.02229*, 2020.

[42] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4pc framework for privacy preserving machine learning," *arXiv preprint arXiv:1912.02631*, 2019.

[43] N. Attrapadung, K. Hamada, D. Ikarashi, R. Kikuchi, T. Matsuda, I. Mishina, H. Morita, and J. C. Schuldt, "Adam in private: Secure and fast training of deep neural networks with adaptive moment estimation," *arXiv preprint arXiv:2106.02203*, 2021.

[44] J.-L. Watson, S. Wagh, and R. A. Popa, "Piranha: A {GPU} platform for secure computation," in *31st USENIX Security Symposium (USENIX Security 22)*, pp. 827–844, 2022.

[45] J. B. Nielsen and C. Orlandi, "Lego for two-party secure computation," in *Theory of Cryptography Conference*, pp. 368–386, Springer, 2009.

[46] J. B. Nielsen, T. Schneider, and R. Trifiletti, "Constant round maliciously secure 2pc with function-independent preprocessing using lego," *Cryptology ePrint Archive*, 2016.

[47] Y. Lindell and B. Pinkas, "A proof of yao's protocol for secure two-party computation. eccc report tr04-063," in *Electronic Colloquium on Computational Complexity (ECCC)*, 2004.

[48] Y. Lindell and B. Pinkas, "A proof of security of yao's protocol for two-party computation," *J. of cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[49] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "Sok: General purpose compilers for secure multi-party computation," in *2019 IEEE symposium on security and privacy (SP)*, pp. 1220–1237, IEEE, 2019.

[50] M. Hashemi, D. Forte, and F. Ganji, "Time is money, friend! timing side-channel attack against garbled circuit constructions," in *International Conference on Applied Cryptography and Network Security*, pp. 325–354, Springer, 2024.

[51] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proc. of the 2012 ACM Conf. on Computer and Comm. security*, pp. 784–796, 2012.

[52] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Intrl. Colloquium on Automata, Languages, and Programming*, pp. 486–498, Springer, 2008.

[53] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, and R. Trifiletti, "On the complexity of additively homomorphic uc commitments," in *Theory of Cryptography Conference*, pp. 542–565, Springer, 2015.

[54] Y. Lindell and B. Riva, "Blazing fast 2pc in the offline/online setting with security for malicious adversaries," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 579–590, 2015.

[55] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Advances in Cryptology-EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings 26*, pp. 52–78, Springer, 2007.

[56] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends® in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018.

[57] R. Zhu, Y. Huang, J. Katz, and A. Shelat, "The {Cut-and-Choose} game and its application to cryptographic protocols," in *25th USENIX Security Symposium (USENIX Security 16)*, pp. 1085–1100, 2016.

[58] M. Keller, E. Orsini, and P. Scholl, "Actively secure ot extension with optimal overhead," in *Annual Cryptology Conference*, pp. 724–741, Springer, 2015.

[59] A. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *arXiv preprint arXiv:1910.12435*, 2019.

[60] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 2020)*, 2020.

[61] M. Ogburn, C. Turner, and P. Dahal, "Homomorphic encryption," *Procedia Computer Science*, vol. 20, pp. 502–509, 2013.

[62] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pp. 409–437, Springer, 2017.

[63] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "Flash: Fast and robust framework for privacy-preserving machine learning," *Cryptology ePrint Archive*, 2019.

[64] A. Patra and A. Suresh, "Blaze: Blazing fast privacy-preserving machine learning." Cryptology ePrint Archive, Paper 2020/042, 2020. https://eprint.iacr.org/2020/042.

[65] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "{SWIFT}: Super-fast and robust {Privacy-Preserving} machine learning," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2651–2668, 2021.

[66] A. Dalskov, D. Escudero, and M. Keller, "Fantastic four:{Honest-Majority}{Four-Party} secure computation with malicious security," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2183–2200, 2021.

[67] C. Orlandi, P. Scholl, and S. Yakoubov, "The rise of paillier: homomorphic secret sharing and public-key silent ot," in *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40*, pp. 678–708, Springer, 2021.

[68] M.-M. Wang, X.-B. Chen, and Y.-X. Yang, "Comment on "high-dimensional deterministic multiparty quantum secret sharing without unitary operations"," *Quantum information processing*, vol. 12, no. 2, pp. 785–792, 2013.

[69] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable byzantine consensus via hardware-assisted secret sharing," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 139–151, 2018.

[70] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "Muse: Secure inference resilient to malicious clients.," in *USENIX Security Symposium*, pp. 2201–2218, 2021.

[71] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "{SIMC}:{ML} inference secure against malicious clients at {Semi-Honest} cost," in *31st USENIX Security Symposium (USENIX Security 22)*, pp. 1361–1378, 2022.

[72] G. Xu, X. Han, T. Zhang, S. Xu, J. Ning, X. Huang, H. Li, and R. H. Deng, "Simc 2.0: Improved secure ml inference against malicious clients," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[73] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable zero knowledge with no trusted setup," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pp. 701–732, Springer, 2019.

[74] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng, "A survey on zero-knowledge proof in blockchain," *IEEE network*, vol. 35, no. 4, pp. 198–205, 2021.

[75] T. K. Frederiksen and J. B. Nielsen, "Fast and maliciously secure two-party computation using the gpu," in *Applied Cryptography and Network Security: 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings 11*, pp. 339–356, Springer, 2013.

[76] Y. Lindell and B. Riva, "Cut-and-choose yao-based secure computation in the online/offline and batch settings," in *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II 34*, pp. 476–494, Springer, 2014.

[77] X. Wang, S. Ranellucci, and J. Katz, "Authenticated garbling and efficient maliciously secure two-party computation," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 21–37, 2017.

[78] B. Kreuter, A. Shelat, and C.-H. Shen, "Billion-Gate secure computation with malicious adversaries," in *21st USENIX Security Symposium (USENIX Security 12)*, pp. 285–300, 2012.

[79] V. Goyal, P. Mohassel, and A. Smith, "Efficient two party and multi party computation against covert adversaries," in *Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27*, pp. 289–306, Springer, 2008.

[80] Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J. Malozemof, "Amortizing garbled circuits," in *Annual Cryptology Conf.*, pp. 458–475, Springer, 2014.

[81] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, and R. Trifiletti, "Tinylego: An interactive garbling scheme for maliciously secure two-party computation," *Cryptology ePrint Archive*, 2015.

[82] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pp. 506–519, 2017.

[83] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, 2017.

[84] L. Song, X. Yu, H.-T. Peng, and K. Narasimhan, "Universal adversarial attacks with natural triggers for text classification," *arXiv preprint arXiv:2005.00174*, 2020.

[85] P. Neekhara, S. Hussain, P. Pandey, S. Dubnov, J. McAuley, and F. Koushanfar, "Universal adversarial perturbations for speech recognition systems," *arXiv preprint arXiv:1905.03828*, 2019.

[86] M. Jagielski, G. Severi, N. Pousette Harger, and A. Oprea, "Subpopulation data poisoning attacks," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3104–3122, 2021.

[87] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[88] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[89] S. Hong, N. Carlini, and A. Kurakin, "Handcrafted backdoors in deep neural networks," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8068–8080, 2022.

[90] S. Goldwasser, M. P. Kim, V. Vaikuntanathan, and O. Zamir, "Planting undetectable backdoors in machine learning models," in *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 931–942, IEEE, 2022.

[91] A. Salem, M. Backes, and Y. Zhang, "Don't trigger me! a triggerless backdoor attack against deep neural networks," *arXiv preprint arXiv:2010.03282*, 2020.

[92] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264, IGI global, 2010.

[93] Y. Ji, X. Zhang, S. Ji, X. Luo, and T. Wang, "Model-reuse attacks on deep learning systems," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 349–363, 2018.

[94] A. Shelat and C.-H. Shen, "Two-output secure computation with malicious adversaries," in *Advances in Cryptology–EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings 30*, pp. 386–405, Springer, 2011.

[95] Y. Lindell, "Fast cut-and-choose-based protocols for malicious and covert adversaries," *J. of Cryptology*, vol. 29, no. 2, pp. 456–490, 2016.

[96] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," *Journal of Cryptology*, vol. 28, no. 2, pp. 312–350, 2015.

[97] S. Hussain, B. Li, F. Koushanfar, and R. Cammarota, "Tinygarble2: Smart, efficient, and scalable yao's garble circuit," in *Proc. of the 2020 WKSP on Privacy-Preserving Machine Learning in Practice*, pp. 65–67, 2020.

[98] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 336–353, IEEE, 2020.

[99] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," *Proceedings on Privacy Enhancing Technologies*, 2019.

[100] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, P. S. Nordholt, and C. Orlandi, "Minilego: Efficient secure two-party computation from general assumptions," in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pp. 537–556, Springer, 2013.

[101] M. Hashemi, S. Roy, D. Forte, and F. Ganji, "Hwgn 2: Side-channel protected nns through secure and private function evaluation," in *Security, Privacy, and Applied Cryptography Engineering: 12th International Conference, SPACE 2022, Jaipur, India, December 9–12, 2022, Proceedings*, pp. 225–248, 2022.

[102] R. Herken, *The universal Turing machine a half-century survey*. Springer-Verlag, 1995.

[103] K.-M. Chung and L. Qian, "Adaptively secure garbling schemes for parallel computations," in *Theory of Cryptography Conference*, pp. 285–310, Springer, 2019.

[104] N. Buescher and S. Katzenbeisser, "Faster secure computation through automatic parallelization," in *24th USENIX security symposium (USENIX security 15)*, pp. 531–546, 2015.

[105] M. Hashemi, S. Roy, D. Forte, and F. Ganji, "Guardianmpc." [Online] https://github.com/esonghori/TinyGarble [Accessed Jan.30, 2023], 2024.

[106] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[107] L. Braun and W. Zakarias, R, "Tinylego framework." [Online]https://github.com/AarhusCrypto/TinyLEGO [Accessed Sep.28, 2023], 2019.

[108] I. Xilinx, "v2021.1." [Online]https://www.xilinx.com/products/design-tools/vivado.html [Accessed: Oct.16, 2024], 2021.

[109] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. of the 2018 on Asia Conf. on computer and Comm. security*, pp. 707–721, 2018.

[110] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "Ezpc: Programmable and efficient secure two-party computation for machine learning," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 496–511, IEEE, 2019.

[111] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[112] S. Imambi, K. B. Prakash, and G. Kanagachidambaresan, "Pytorch," *Programming with TensorFlow: solution for edge computing applications*, pp. 87–104, 2021.

[113] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pp. 113–124, 2011.

[114] C. Aicher, N. J. Foti, and E. B. Fox, "Adaptively truncating backpropagation through time to control gradient bias," in *Uncertainty in Artificial Intelligence*, pp. 799–808, PMLR, 2020.

[115] B. Pulido-Gaytan, A. Tchernykh, J. M. Cortés-Mendoza, M. Babenko, G. Radchenko, A. Avetisyan, and A. Y. Drozdov, "Privacy-preserving neural networks with homomorphic encryption: C hallenges and opportunities," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1666–1691, 2021.

[116] F. Bourse, R. D. Pino, M. Minelli, and H. Wee, "Fhe circuit privacy almost for free," in *Annual Intrl. Cryptology Conf.*, pp. 62–89, Springer, 2016.

[117] C. Gentry, *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[118] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 707–723, IEEE, 2019.

[119] A. Oprea and A. Vassilev, "Adversarial machine learning: A taxonomy and terminology of attacks and mitigations," tech. rep., National Institute of Standards and Technology, 2023.

[120] X. Huang, M. Alzantot, and M. Srivastava, "Neuroninspect: Detecting backdoors in neural networks via output explanations," *arXiv preprint arXiv:1911.07399*, 2019.

[121] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks.," in *IJCAI*, vol. 2, p. 8, 2019.

[122] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li, "Detecting ai trojans using meta neural analysis," in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 103–120, IEEE, 2021.

[123] G. Kolhe, T. Sheaves, K. I. Gubbi, T. Kadale, S. Rafatirad, S. M. PD, A. Sasan, H. Mahmoodi, and H. Homayoun, "Silicon validation of lut-based logic-locked ip cores," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1189–1194, 2022.

[124] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of sat instances for obfuscating circuits using fully configurable logic and routing blocks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[125] K. I. Gubbi, I. Kaur, A. Hashem, S. M. PD, H. Homayoun, A. Sasan, and S. Salehi, "Securing ai hardware: Challenges in detecting and mitigating hardware trojans in ml accelerators," in *2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 821–825, IEEE, 2023.

[126] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *Proceedings of the 52nd Annual Design Automation Conference*, pp. 1–6, 2015.

[127] E. M. El Mhamdi, R. Guerraoui, and S. L. A. Rouault, "Distributed momentum for byzantine-resilient stochastic gradient descent," in *9th International Conference on Learning Representations (ICLR)*, 2021.

[128] A. ARM, "Architecture reference manual," *ARMv7-A and ARMv7-R edition*, 2012.