IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.DOI

GuardianMPC: A Backdoor-resilient Deep Learning Accelerator

MOHAMMAD HASHEMI[1], (Member, IEEE), DOMENIC J. FORTE[2], AND FATEMEH GANJI[1], (Member, IEEE)

[1]Electrical and Computer Engineering Department, Worcester Polytechnic Institute, Worcester, MA 01609 USA (e-mail: mhashemi@wpi.edu, fganji@wpi.edu)
[2]Electrical and Computer Engineering Department, University of Florida, Gainesville, FL 32611-6200 USA (email: dforte@ece.ufl.edu)

Corresponding author: Mohammad Hashemi (e-mail: mhashemi@wpi.edu).

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

ABSTRACT Advancements in deep learning (DL) have unlocked unprecedented capabilities in various domains. However, alongside these advancements, the susceptibility of DL models to backdoor insertion attacks has become a significant concern. These attacks mimic the actions of a malicious adversary and can remain undetected under conventional security paradigms. MPC's interactive computation approach aligns well with the need to counter backdoor attacks. In this work, we aim to build a bridge between MPC, and particularly, secure and private two-party computation, and the countermeasures against backdoor attacks. In doing so, we introduce GurdianMPC, a DL accelerator that employ the LEGO family of protocols to provide not only security against backdoor attacks, but also users with data privacy. Moreover, by leveraging optimization techniques tailored toward hardware acceleration, GurdianMPC achieves high efficiency and low latency. Specifically, our implementation on Field Programmable Gate Arrays (FPGAs) computational power evaluate a typical NN up to $13.44\times$ faster during the online phase compared to its counterpart running on software, with a trade-off against a negligible offline computation time.

INDEX TERMS Backdoor Insertion, Malicious adversary, Deep learning, Secure function evaluation, Private function evaluation.

I. INTRODUCTION

Deep learning (DL) has seen remarkable progress in revolutionizing areas such as image and speech recognition and 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 feature 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) using a pre-trained model. Here, outsourcing 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 a market [6]–[8]. In this paper, we focus on outsourcing and using pre-trained models, where the reason is twofold. First, outsourcing has the highest attack's success rate since the attacker has access to the model, training data, and controls the training process. On the contrary, the user has restricted computational resources, i.e., the reason behind why it has been decided to rely on outsourcing. The same limitation holds for 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 of such attacks is the stage of the 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) weight-based backdoors inserted 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[1]. 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, increasing deployment of NNs in various applications has also led to raising privacy concerns. As a remedy, secure inference protocols have been devised, where many of them have secure multiparty computation (MPC), specifically, secure two-party computation at their core [17]–[32]. These protocols protect the user's data and/or NN providers' intellectual property (IP), i.e., the NN models. Now the question is whether backdoor attacks can be formalized within the context of MPC. To answer this question, we should elaborate on adversary models defined in the realm of MPC.

Adversaries as seen by MPC. In this regard, 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. This clarifies why backdoor attacks can be formalized in the malicious adversary model. 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. Still, the evaluator has no way to confirm the circuit is correct [33].

Garbled circuits. Among all the approaches presented in MPC domain, garbled circuit (GC) -based approaches have emerged as a successful technique in providing robust protection against malicious adversaries. GCs are ordinary circuit, but each wire carries a string-valued token rather than the bit it represents. GCs are often realized by encoding the parties' inputs and sending them to an evaluator during an online phase, whereby revealing no information about the inputs. Given that the NN model is one party's input, as considered in private function evaluation (PFE), the protection offered by GCs can be a good fit to counter backdoor insertion attacks in NNs.

Contributions. Our paper aims to answer the following questions on application of PFE in the protection of NNs against backdoor attack. in view of the fact that GC-based NN inference has attracted a great deal of attention, which inference engines are immune against backdoor attacks, especially attacks mounted during outsourcing and against pretrained models? If existing GC-based NN inference engines are not secure against such attacks, which modifications can be made to assure their security? How much cost does making GC-based NN inference engines secure against backdoor attacks impose?

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

1) Our observation is that only a limited number of GC-based inference engines are secure against backdoor attacks. This is concluded by building a bridge between the definitions of malicious adversaries in MPC and backdoor attacks. To narrow this gap, we introduce GuardianMPC, a novel NN inference engine that is secure against backdoor attacks.
2) GuardianMPC greatly benefits from the flexibility offered by LEGO protocol family [34], in particular, one of its variant that enables function-independent preprocessing [35]. This feature allows us to protect the NN model (architecture and parameters) efficiently. Furthermore, this facilitates GC-based ineference through optimizations and parallelism provided by both the protocol and Field Programmable Gate Arrays (FPGAs).
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 NN model as a private input of the parties. A slight increase in protocol's pre-processing time (in the offline phase) is traded off against the security in the malicious case.
4) GuardianMPC is secure in the presence of malicious adversary with an online performance close to the cutting-edge HbC-resilient DL accelerator.

Organisation: Section III includes the preliminaries and describes a taxonomy of the existing relevant malicious adversary-resilient GC-based frameworks. Section IV elaborates on the relationship between backdoor attacks and malicious adversary as defined in the context of MPC. Section V details methodology, whereas Section VI includes experimental results for real-world DL model implementation using GuardianMPC. Section VII discusses the important points relevant to this work. Section VIII, finally concludes the paper.

II. RELATED WORK

A. NN COMPUTATION WITH MALICIOUS ADVERSARY

As explained in Section I, secure NN computation can be solved 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. Such adversaries can cheat and arbitrarily deviate from the defined protocol, endangering the privacy and security of computations [36]. In fact, GC-based 2PC protocols can be turned to maliciously secure ones by employing the combination of cut-and-choose techniques [37] and malicious OT-extension [38]. To tackle the large overhead imposed by such combinations, various techniques have been devised, 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

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

protocols for NN inference. Nevertheless, among them, only a few proposals guarantees security in the malicious sense, for instance, [39]. One possible reason is that HE as an encryption method suffers from limited operational scope, potential truncation errors, complex key management, and compatibility issues [40]–[42]. 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.

Secret sharing is another ingredient often used in protocols offering security against malicious adversary. Examples of this are [43] and its most related protocols FLASH [44], BLAZE [45], Falcon [46], Trident [47], SWIFT [48], Adam in Private [49] and Fantastic 4 [50]. In addition to supporting more parties (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 [44], SWIFT [48], Trident [47] 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, makes sure that honest parties always receive the correct computation result cf. [51]. Another aspect making these protocols different is how the protocol is optimized. For instance, SWIFT [48] aims for high-speed and robust privacy-preservation in machine learning by employing algorithmic and computational optimizations. Needless to say that secret sharing adds computational complexity, poses vulnerability to collusion attacks, and presents challenges in managing and distributing shares [52]–[54].

Using zero knowledge proofs. Primitives relying on zero knowledge (ZK) proofs allow the NN model owner to convince the users that the predictions are correctly computed using the NN model. In this context, ZK proofs guarantee that if the model owner sends a wrong result of the computation, it can only pass the verification with a negligible probability, which is 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., [45]. Lehmkuhl et al. [55] introduced MUSE, a secure machine learning inference framework against malicious clients, leveraging HE and secret sharing MPC and zero-knowledge proofs. SIMC [56] and SIMC 2.0 [57] have further enhanced the efficiency of MUSE by building upon its protocols including ZK proofs. Beside the issue mentioned in regard to secret-sharing and HE, one should not ignore the difficulties facing the adoption of zero-knowledge proofs. Despite its performance efficiency, zero-knowledge proofs can introduce computational complexity, require trusted setups, and may face scalability issues [58], [59].

B. PURE GC-BASED APPROACHES

As briefly explained before, pure GC protocols in HbC setting have a clear path towards supporting malicious security; nevertheless, this can be achieved at the cost of high overhead. Nonetheless, main technique for securing GC protocols against malicious adversaries is cut-and-choose [35], [60]–[64]. 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 verify whether all the circuits have been generated correctly. If so, the user continues running the protocol and running 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 1 summarizes some of these techniques, their target attacks, and 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. [64] 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. [65] 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 through revealing the random seeds used to produce them rather than the check circuits themselves as suggested in [66]. Frederiksen et al. [61] 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.

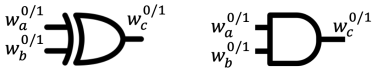
Amortized cut-and-choose has been proposed to reduce the evaluation cost in a scenario, where two parties know in advance that multiple secure evaluations of the same function (on unrelated inputs) should be performed. In this setting, the amortized costs for each evaluation can be reduced by performing a single cut-and-choose for all evaluation instances cf. [33]. This idea has been formalized, later optimized and implemented in [60], [62], [67]. Lindell et al. [62] 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. Further, Nielson et al. [35] expanded upon these concepts by implementing function-independent preprocessing, leading to constant-round maliciously secure 2PC.

III. 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 secure function evaluation (SFE) method for Boolean

TABLE 1: Summary of most relevant garbled-circuit-based countermeasure against malicious adversary.

Approach	Technique	Mitigation	Contributions
Lindell et al. [64]	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. • Consistency checks based on cut-and-choose.
Kreuter et al. [65]	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. [61]	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. [67]	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. [62]	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 [35]	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. [60]	ZK Proofs + GC	• Selective-OT. • Incorrectly construct GC. • Input consistency.	• Optimizing the online/offline phase of [62] by reducing the number of GC needed to be evaluated during the online phase



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	$E_{w_a^1,w_b^1}(w_c^1)$

FIGURE 1: Garbled gates look-up table (Inspired by [71]).

circuits [68]–[70]. 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 generate a garbled table T_i per logic gate G_i, where in each row the output is encrypted in relation to the tokens. 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 scrambled 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 [72]. It is also possible to skip the garbling of the circuit's output wires, allowing both parties to determine solely the final result [73].

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 [73].

IV. DL AND GC: SIMILARITIES IN THREAT MODELS

After reviewing the related work and the background on GC, this section aims to link the adversary models defined in the context of MPC and backdoor attacks. Figure 2 shows the well-known attack types against each stage of the DL pipeline. According to Figure 2, some well-known attack types against DL pipeline are adversarial example attack [74], universal adversarial patch [75]–[77], data poisoning [78], [79], backdoor insertion [79]–[82], and outsourcing [80][2]. Among these attacks, backdoor insertion attacks [79]–[82] effectively reconfigure neural networks to perform undesired actions. Moreover, backdoor insertion can directly perform on the hardware implementations of the DL accelerator, which is more relevant to the scope of this work. A brief summary of these attacks is provided in the following.

[2]Note that data poisoning/adversarial machine learning are beyond the scope of this work.

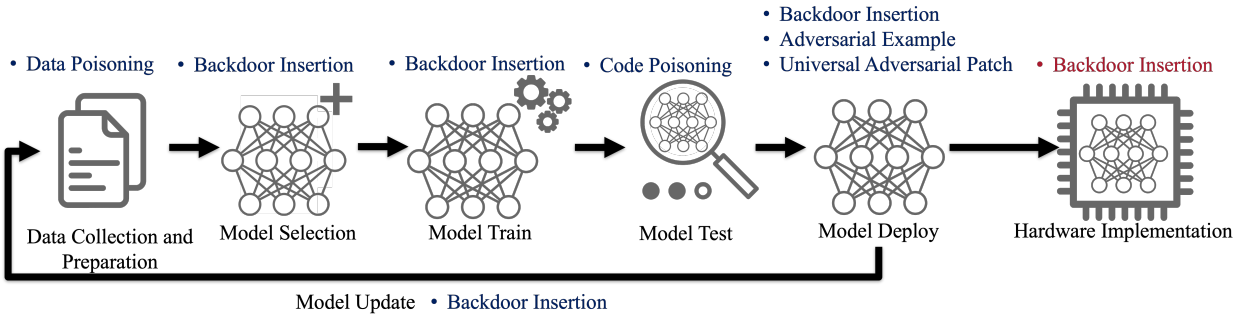


FIGURE 2: Well-known attack types against each stage of the DL pipeline (Inspired by [8]). The red font means that the attack is mounted on hardware implementation of the DL model. The backdoor insertion during different phases involves architectural backdoor insertion in *Model Selection*, direct weight manipulation in *Model Train*, architectural backdoor insertion and direct weight manipulation in *Model Deploy*, and direct weight manipulation in *Model Update*.

A. BACKDOOR ATTACKS IN DL PIPELINE

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

1) 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 can modify the weights of the trained model, so-called direct weight manipulation attacks [81], [82], while maintaining the accuracy of the model. In such a scenario, the model appears to function correctly under normal operational tests since the backdoor only activates in the presence of the predefined trigger, making the backdoor difficult to detect.

Architectural backdoor insertion attack. Architectural backdoor insertion attacks subtly embed a backdoor into an ML model, enabling it to handle both a malicious sub-task and a benign primary task [79], [80]. It reduces the model's accuracy even if it is retained on clean inputs [80]. This makes detection challenging since they mirror the behavior of unaltered models on standard tests. However, upon encountering a concealed 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 insertion attack is the hardware backdoor insertion [14]. This involves embedding backdoors within the architecture of a neural network. Such alterations can be done by modifying the gates in the circuit or fundamental building blocks of the neural network's structure [14].

2) Pretrained model backdoor insertion

Pretrained 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 [83], having been trained on large, generalized datasets. In this scenario, the backdoor is introduced by either direct weight manipulation attacks [81], [84] or by altering an existing model to include the backdoor [14], [79], [80]. 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 attack that can be mounted on pre-trained models is referred to as direct weight manipulation. Such attacks involve partially/fully altering a pre-trained neural network's weights to introduce a backdoor without the need to poison the training data [81], [82]. This approach offers more control over the model's behavior, as it directly changes model parameters in a way that can evade many existing backdoor detection and removal defenses [81]. These attacks can maintain high success rates of triggering malicious behavior while preserving the model's overall performance on legitimate tasks. The modification process is tailored to bypass defensive strategies and can be highly efficient, requiring only a small number of samples to be effective.

B. TARGETS OF MALICIOUS ADVERSARY IN GC

What can be understood from the discussion in Section IV 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 between these threats and attacks against MPC protocols has been unnoticed. By linking the similarities between attacks on DL pipelines and those on MPC, we aim to identify shared vulnerabilities and develop more robust defense strategies through private evaluation of GCs (i.e., PFE). Hence, be-

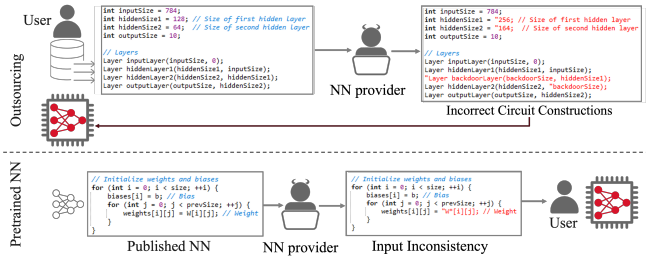


FIGURE 3: An example of most relevant malicious activities as known in MPC and their possibility in the DL pipeline.

fore enumerating the similarities between the attacks on DL pipelines and those on MPC, we briefly describe relevant attacks in the sense of MPC.

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, in protocols relying on cut-and-choose [62], e.g., LEGO protocols [34], multiple GCs are prepared, and a subset is chosen randomly for evaluation. 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 NN provider or the user applies different inputs in different computation instances cf. [85]. Suppose that the NN provider provides 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 something about the user's input based on whether or not it aborts. This undermines the security and reliability of the protocol [86].

C. SIMILARITIES BETWEEN ADVERSARIES

Figure 3 shows an example of where and how incorrect circuit construction and input inconsistency attacks can be launched by a malicious party in an MPC framework that involves an NN provider and a user. In such a scenario, the parameters and hyperparameters of the NN, i.e., weights, bias and model architecture are considered the NN provider's secrets. Both incorrect circuit construction and input inconsistency attacks can be launched by a malicious NN provider.

Direct weight manipulation [81], [82] changes the expected behavior of a DL model by altering its trained weights, effectively planting a backdoor that can bypass detection mechanisms. This is a case for constructing a garbled circuit that computes a function that is different to the one that two parties agreed to compute cf. [64], [87]. 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 [86] in GCs. Moreover, architectural backdoor insertion [8], [14], [80] along with the direct weight manipulation are two possible attacks to be launched during outsourcing the NN computation. Architectural backdoor insertion aims to modify the DL model through its structure without affecting the model functionality. 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 [62].

D. OUR ADVERSARY MODEL

Our model involves two parties, a user and an NN provider. The user aims to obtain an NN for a certain task, whereas the NN provider represent 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 checks the accuracy of the trained model on a held-out validation dataset. When performing this process in an interactive manner, the NN provider may change the weights after validation phase is over. Another attack vector in this case concerns making changes to the NN requested by the user. When using a pretrained NN, the user downloads a maliciously pre-trained model crafted by the NN provider, who first downloaded 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 modify the NN in an arbitrary manner.

As explained in Section IV-C, the adversary in these cases can be categorized as a malicious adversary in the GC sense. Under these backdoor attack scenarios, the malicious NN provider can create a trained network with state-of-the-art performance on the user's training and validation samples, which behaves poorly on specific attacker-chosen inputs, referred to as the backdoor trigger [6], [8], [80], [88]. In order to launch both direct weight manipulation and architectural backdoor attacks [8], [14], [80] against hardware NN accelerators, the adversary can make changes at the gate or register-transfer level (RTL) of the circuit.

V. GUARDIANMPC

Our framework, GuardianMPC, aims to stop attackers from mounting architectural backdoor and direct weight manipulation attacks. As explained in Section IV-C, such attacks are

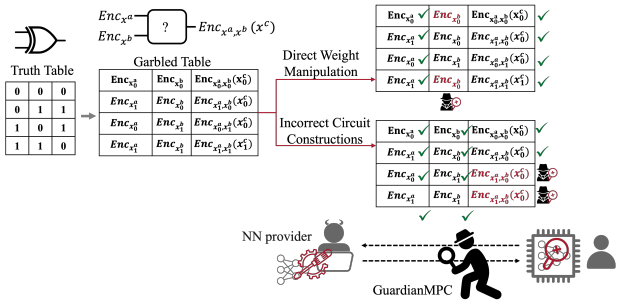


FIGURE 4: How GuardianMPC protects DL against the malicious NN owner during outsourcing.

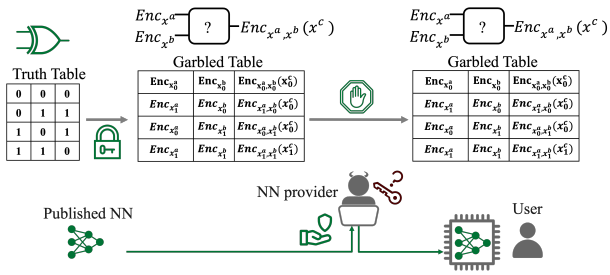


FIGURE 5: How GuardianMPC ensures privacy of pretrained NNs in the face of backdoor attack.

in line with incorrect circuit construction and input inconsistency as formalized within the framework of maliciously-secure GCs. Both the direct weight manipulation and architectural backdoor attacks [8], [14], [80] fundamentally rely on changes at the gate or Register-Transfer Level (RTL) level of the circuit when considering hardware accelerators. In these cases, the underlying structure or logic of the system is altered to introduce vulnerabilities or hidden functionalities. Therefore, if manipulation of the DL accelerator structure or logic of the system can be prevented, launching a backdoor insertion attack against DL accelerators can be mitigated. Moreover, as the success of the direct weight manipulation attacks [81], [82] depends on altering the NN's weights, encoding the weights stops any modification to the weights in pretrained models.

These are aligned with protecting a circuit from a malicious adversary who can change the circuit, providing inconsistent inputs to the different GCs, and, in general, deviation from the protocol execution. According to the above-mentioned arguments, we have introduced GuardianMPC based on the LEGO [34] scheme to prevent backdoor attack insertion in architecture and change the hyper-parameters of the DL interfaces [8], [14], [80], [89]–[91], or a direct weights manipulation [81], [82] by a malicious NN provider.

A. WHY LEGO CAN PROTECT DL AGAINST BACKDOOR ATTACKS

The main contribution of large efficient garbled-circuit optimization (LEGO) protocol [34] is its ability to allow the efficient construction of GCs that are secure against malicious adversaries without the substantial overhead typically associated with providing such security. The protocol uses a novel approach where a batch (multiple pairs of inputs) of independently garbled NAND gates, rather than entire garbled circuits, is evaluated through cut-and-choose. From the set of garbled NAND gates, the evaluator can check a fraction of it for correctness. The remaining unopened gates are assigned into buckets, assembled into a garbled circuit in a process called "soldering," [34], [92]. Finally, the evaluator evaluates the single conceptual garbled circuit, guaranteed to behave like a correct garbled circuit with high probability.

TinyLEGO Frederiksen et al. [92] introduced TinyLEGO based on the LEGO [34] scheme for two-party computation (2PC) designed to be secure against malicious adversaries. In a 2PC protocol, two parties wish to compute a function over their joint inputs without revealing their individual inputs to each other. The TinyLEGO scheme does this by "garbling" the circuits that represent the computation, meaning it transforms them into an encoded form where the actual computation can be carried out without either party seeing the actual values at any step.

While the LEGO [34] scheme was not explicitly designed for securing deep learning interfaces, its principles of interactive garbled computations can offer a unique approach to detecting and mitigating backdoor attacks. Any malicious alterations to the deep learning circuits can be detected and prevented by ensuring transparency, verifiability, and interactive participation in the computation. The following arguments support the claim that any framework that is built based on the LEGO [34] scheme can prevent backdoor attack insertion in architecture or change the hyper-parameters of DL interfaces by a malicious provider.

Transparency in computation By using an interactive garbling scheme like LEGO [34], it is possible to ensure that every computation (in this case, every forward and backward pass during training or inference in a neural network) is done in a transparent and verifiable manner [34]. Any unauthorized change or a backdoor in the circuit would disrupt this garbling process.

Protection against unauthorized changes If a malicious actor attempts to introduce a backdoor into the model by altering its circuitry (the neural network's architecture or weights), the garbled computation would not produce the expected results unless both parties (the user and the service) are aware of and agree to the changes [72] This provides a check against unauthorized alterations.

Secure two-party computation In scenarios where two entities (e.g., a user and a cloud service) are jointly executing a deep learning task without wanting to reveal their data or model details to each other, the LEGO [34] scheme ensures that neither party can maliciously alter the deep learning circuit without detection.

Detection of malicious behavior Even if a backdoor were previously embedded in the deep learning model, any attempt to trigger it during a garbled computation would result in unexpected behavior, making the backdoor evident [60].

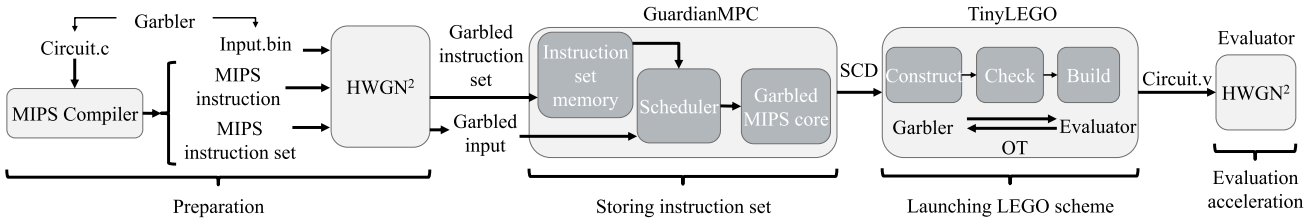


FIGURE 6: A high level flow of GuardianMPC.

User's control over circuitry Given the interactive nature of the LEGO [34] scheme, users can be actively involved in the computation process, allowing them to detect and prevent potential backdoor triggers.

B. FLOW OF GUARDIANMPC

Figure 6 illustrates a high-level flow of GuardianMPC. The flow of GuardianMPC can be described in four phases as follows:

Preparation TinyLEGO [93] is the heart of GuardianMPC. Similar to some of the prior frameworks in MPC [94]–[96], TinyLEGO accepts Simple Circuit Description (SCD) as its input. Although this type of file, SCD, is used in the flow of GuardianMPC, the input to the GuardianMPC is the MIPS instruction, which is different from the TinyLEGO framework.

Hence, as the first step of GuardianMPC flow, preparation, the garbler compiles the circuit function, `Circuit.C`, and generates the corresponding MIPS instructions, `MIPS instruction`, which act as the garbler's input, and the `MIPS instruction set`, which act as the op-code mapping in the MIPS architecture, of the circuit. The `MIPS instruction`, `MIPS instruction set`, and the `Input.bin` then are provided to HWGN^2. After that, HWGN^2 generates the `Garbled input` and `Garbled instruction set`, which will be used in the next step of GuardianMPC, Storing instruction set, to generate the `Garbled MIPS core`. For a detailed explanation of this phase, refer to HWGN^2 [96].

Storing instruction set After the corresponding MIPS instructions, `Garbled input`, and the MIPS instruction set are generated and garbled, the garbler stores them in the instruction set memory of GuardianMPC. The `Garbled instruction set` and `Garbled input` storage have two benefits for GuardianMPC.

First, the storage of the `Garbled instruction set` into the GuardianMPC `Instruction set memory` allows the `Scheduler` to access and decode the instructions into the processor operation inside the core, inside the chip, instead of accessing external memory, outside the chip, which is one of the most important bottlenecks of the processor performance.

Secondly, the ability to access the `Garbled input` removes the requirement of the connection between the garbler and the 3PIP provider, in the case of pretrained NN explained in Section IV-A2, which is one of the major costs of maliciously secure MPC frameworks.

It should be noted that `Garbled instruction set` and `Garbled input` are encrypted, garbled, by the garbler and only the garbler can decrypt them, as only the garbler knows the key based on which `Garbled instruction set` and `Garbled input` are encrypted. Therefore, the storage of `Garbled instruction set` inside the `Instruction set memory` and `Garbled input` inside the `Scheduler` allows not the user nor the 3PIP provider to reveal the garbler secrets.

At the end of this phase, the `Garbled MIPS core` including an `Instruction set memory`, which holds the `Garbled instruction set`, and the `Scheduler`, which has the `Garbled input`, is converted to the SCD format, see HWGN^2 [96] for the details of this conversion. This SCD format is ready to be processed for ensuring security and privacy against malicious adversaries through the LEGO [34] protocol.

Launching LEGO scheme At this phase, GuardianMPC utilizes the LEGO [34] protocol to check the input consistency of the NN provider and if the NN model is constructed correctly. This checking process involves in three phases: (1) `Construct`, (2) `Check`, and (3) `Build`. These phases are taken care of by TinyLEGO [93] framework during the GuardianMPC flow.

At the first step, `Construct` the GuardianMPC utilizes Tinyconst presented by TinyLEGO [93] to construct many instances of the GC of the `Garbled MIPS core` based on the `SCD`, which includes the `Garbled MIPS core` execution module, the `Garbled MIPS instruction set` stored on `Instruction memory set`, and `Garbled input` stored in the `Scheduler`. After the construction, the garbler and evaluator engage in LEGO [34] protocol, as the second phase of the execution of the Tinyconst, to complete the `Check` phase. If the evaluator ensures that most instances of the GC are constructed correctly, it builds the GC, `Build` phase, from the unchecked instance. At the end of the `Build` phase, the Circuit.v is generated, which is the protected form of `Garbled MIPS core` against Incorrect circuit constructions and Input inconsistency. After that, the `Garbled MIPS core` is ready to be evaluated.

Evaluation acceleration The evaluation phase of GuardianMPC is taken care of by the garbled MIPS evaluator presented in [96]. A garbled MIPS evaluator is a hardware-

based secure MIPS core accelerator, which is replaced by the software-based TinyEval [93] during the flow of GuardianMPC. The replacement of the garbled MIPS evaluator with the TinyEval enhances the performance of the evaluation phase, which is a part of the GuardianMPC online phase. Moreover, as the MIPS instructions are garbled during their execution by the Garbled MIPS evaluator, the functionality of the circuit is kept hidden from the 3PIP provider and user, which can prevent the 3PIP provider from backdoor insertion in the NN model or the user from launching a successful attack to reveal the NN model weights, attack such as [97]–[99].

During the evaluation phase, both the garbler and the evaluator communicate over OT, to exchange their inputs. After the OT, the evaluator has his garbled input and can evaluate the `Circuit.v`, following the flow of HWGN^2 [96], to receive the garbled output. As the final step of the evaluation, the evaluator decrypts the garbled output and finds the `Circuit.v` output corresponding to his input.

It should be noted that during the evaluation phase, the evaluator cannot decrypt the `Garbled MIPS instruction set` or `Garbled input`, embedded in the `Circuit.v` as the evaluator does not have the key based on which these secrets are garbled; therefore, the security of the garbler's input is preserved. At the end of the evaluation phase, the garbler sends the output decryption labels to the evaluator, and the evaluator can decrypt the garbled output.

C. DIFFERENCES BETWEEN GUARDIANMPC AND TINYLEGO

Although both TinyLEGO [92] and GuardianMPC are built upon LEGO [34] scheme, they have some differences as follows.

Underlying architecture GuardianMPC primarily focuses on translating the function into a sequence of low-level machine instructions using the MIPS architecture. This abstraction allows for hiding the function's exact operations and logic. On the other hand, TinyLEGO operates at a higher level, utilizing garbled gates to construct circuits without delving into low-level machine instructions. This fundamental architectural difference sets the stage for the varying privacy mechanisms in the two systems.

Function privacy In GuardianMPC, converting a function's functionality into MIPS instructions can serve as a form of obfuscation. While the evaluator has access to these instructions, discerning the original high-level logic or purpose becomes a non-trivial task, thereby ensuring the privacy of the function in the concept of secure multi-party computation (SFE). Contrarily, the TinyLEGO system mandates both parties to know the function, represented as a circuit. Here, the emphasis is on keeping the inputs hidden, but the structure and logic of the function remain transparent.

Applicability and overhead The overhead introduced in GuardianMPC due to the conversion into MIPS instructions is a trade-off for enhanced function logic privacy. More complex functions could lead to extensive MIPS instruction sets in this paradigm, potentially impacting efficiency. Conversely, TinyLEGO emphasizes efficiency by reusing garbled gates, which reduces the overhead inherent in garbling a circuit. This efficiency in TinyLEGO, however, comes at the price of revealing the function's structure.

Flexibility and complexity The specificity of the MIPS instruction set in GuardianMPC might pose challenges in representing diverse functions. Some functions might either be harder to represent or might necessitate an expansive set of instructions. In stark contrast, TinyLEGO's abstract representation based on circuits offers more flexibility in capturing a wide range of functions. Yet, this flexibility in TinyLEGO comes tethered with the requirement for both parties to possess knowledge of the function's specifics.

The difference between GuardianMPC and the TinyLEGO approach relies on balancing function privacy and computational overhead. GuardianMPC gravitates towards preserving function privacy, making it challenging for evaluators to decrypt the original function. While accentuating efficiency, TinyLEGO demands mutual cognizance of the function's logic between both parties. The choice between the two methodologies hinges squarely on the privacy and efficiency benchmarks set by the application in question.

D. THE IMPORTANCE OF FUNCTION SECRECY IN DL INTERFACE

In the realm of secure function computation (SFE), particularly in the application of DL accelerators, the concept of function privacy is crucial for mitigating sophisticated threats like backdoor attacks. GurdianMPC, as detailed in your paper, exemplifies this principle by converting a function's functionality into MIPS instructions, which serves as an effective form of obfuscation. This approach is a key differentiator from frameworks like TinyLEGO [93].

In GurdianMPC, while the evaluator has access to the MIPS instructions, unraveling the original high-level logic or purpose from these instructions is a complex task. This complexity ensures the privacy of the function, making it difficult for an adversary to decipher or manipulate the underlying logic of the DL models. This obfuscation of the function's core operations, while still allowing for its computational execution, provides a robust layer of security in the SFE context.

Conversely, cutting-edge frameworks like TinyLEGO [93] operate differently. They require both parties involved in the computation to have knowledge of the function, which is represented as a circuit. In such frameworks, the focus is on keeping the inputs hidden, but the structure and logic of the function remain transparent. While beneficial in certain scenarios, this transparency of the function's logic and structure in such frameworks can be a vulnerability when protecting against more sophisticated threats like backdoor attacks by a third-party provider or during the transfer learning techniques [8].

VI. EXPERIMENTAL RESULTS

TABLE 2: The resiliency of GC-interfaces against backdoor insertion during the transfer learning, Pre-trained backdoor insertions, or directly to the model during outsource interfaces, according to [100].

Framework	Outsourced Backdoor	Pretrained Backdoor
ABY^3 [43]	Y	N
TinyGarble2 [101]	N	Y
CryptFlow [102]	N	Y
Flash [44]	N	Y
Blaze [60]	N	Y
Swift [48]	N	Y
Trident [47]	Y	N
Fantastic 4 [50]	Y	N
QuantizedNN [39]	N	Y
MUSE [55]	N	Y
AdamInPrivate [49]	Y	N
SecureNN [103]	Y	N
FalcoN [46]	Y	N
GuardianMPC	Y	Y

A. EXPERIMENTAL SETUP

We assessed the execution time of benchmarks using Intel Xeon Silver 16 core CPU @$2.5GHz$, NVIDIA $RTX-A4000$ GPU, 128 GB RAM, and Linux Ubuntu 20 as the garbler, and a system equipped with an Intel Core $i7-7700$ CPU @$3.60GHz$, 16 GB RAM, and Linux Ubuntu 20 as the host CPU of the evaluator who established the connection between FPGA and the garbler. The hardware implementation of benchmarks has been implemented on Artix-7 FPGA device $XC7AT100T$ with package number $FTG256$ using Xilinx Vivado Design Suite 2021 [104]. To have a fair comparison, we have implemented the benchmarks using three methods as follows:

- The first case is the implementation of benchmarks, using Xilinx Vivado Design Suite 2021 [104], without any protection such as a garbling scheme or conversion to MIPS instructions (hereafter called **No Protection, no Hiding**).
- The second case is the implementation of benchmarks using TinyLEGO [93] on software (**Protected, no Hiding**).
- The third case is the implementation of benchmarks using GuardianMPC (**Protected, with Hiding**).

B. CANDIDATE BENCHMARKS

Multi-layer Perceptron As the first candidate benchmark, we focus on a tiny multi-layer perceptron (MLP) used by [96] with 784 neurons in its input layer, three hidden layers, each with 1024 neurons, and an output layer with 10 neurons that are trained on MNIST (hereafter called **BM1**). The second MLP candidate is used by SecureML [18] and MiniONN [20] (hereafter called **BM2**). It has 784 neurons in its input layer, one fully connected hidden layer with 128 neurons and square activation function (AF), and an output layer with 10 neurons trained on the MNIST dataset [20].

Convolutional Neural Network We have also implemented Convolutional Neural Networks (CNN) in this work as the heavier computation cost requirement candidate benchmarks using three cases explained in Section VI-A with the purpose of the scalability of our work evaluation. The first CNN we implemented has been considered by Chameleon [105], EzPC [106], and MiniONN [20]. It has seven convolutional layers, each followed by a ReLU AF, input size of $3\times32\times32$, and output size 10 trained on the CIFAR-10 dataset (hereafter called **BM3**), for a detailed configuration of this CNN, refer to [105]. The second CNN, LeNET5 [107], has been considered by TinyGarble2 [101], which includes a convolutional layer with 6 filters of 5×5 kernel size, using Tanh as the AF. This is followed by a pooling layer featuring 2×2 average pooling. The next layer is another convolutional layer, with 16 filters, 5×5, and again using Tanh. Following this is another pooling layer with 2×2 average pooling. The network then includes two fully connected layers with 120 and 84 neurons, both employing Tanh as the AF. Finally, the output layer uses a Softmax activation function.

C. EXPERIMENTAL RESULTS

BM1 Table 3 contains the execution time cost results comparison between each phase of GuardianMPC, TinyLEGO [92], and the BM1 without any protection and hiding. According to Table 3, the online phase execution time cost, including checking, building, and evaluation phase, of BM1 benchmark implementation using TinyLEGO [93] is $149\times$ than the FPGA-based accelerator implementation of BM1, without any protection and hiding. This decrease in performance is the cost of securing the computation against a malicious garbler and ensuring the evaluator's input's privacy. Moreover, it is observable in Table 3 that the implementation of BM1 using GuardianMPC costs a total of 58% more execution time in the pre-processing phase and 20% more execution time in the online phase compared to the implementation of BM1 using TinyLEGO [93].

The extra execution time cost comes with the benefit of securing the circuit function as discussed in Section V-C. Although using GuardianMPC to implement benchmarks, such as BM1, results in a decrease in performance, it hides the function from the evaluator, which prevents an adversarial evaluator from launching side-channel attack similar to [97] against the benchmarks as described in [96].

BM2 Table 4 contains the execution time cost results comparison between the online and offline phase of GuardianMPC, TinyLEGO [92], MINIONN [20], and the BM2 without any protection and hiding. According to Table 4, the online phase execution time cost of BM2 benchmark implementation using TinyLEGO [93] and GuardianMPC is $\times2.3$ and $\times2.48$ more than the FPGA-based accelerator implementation of BM2, without any protection and hiding.

Compared to MINIONN [20], GuardianMPC requires 6% more online time to execute BM2. It should be mentioned that we have not implemented the Single Instruction, Multiple Data (SIMD) technique on TinyLEGO [108] part of our work. Moreover, GurdianMPC provides resiliency against a stronger adversary model, malicious adversary, while MINIONN [20] is resilient against a semi-honest adversary model. However, the online execution time of GuardianMPC

TABLE 3: Execution time cost comparison between each phase of BM1 implementation using GuardianMPC, TinyLEGO [92], and without any protection and hiding.

Protection, Security	# of AND gates	Preprocessing (ms)				Online (ms)			
		Construction	BaseOT	Random Generation	**Total**	Checking	Building	Evaluation	**Total**
No Protection, no Hiding	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	**3.43**
Protected, no Hiding (TinyLEGO [93])	2098	3591.8	579.03	17.59	4188.42	492.7	6.09	43.84	542.63
Protected, with Hiding (GuardianMPC)	**2098**	**5912.29**	**719.88**	**18.12**	**6650.29**	**608.94**	**6.23**	**3.26**	**618.43**

TABLE 4: Execution time cost comparison between each phase of implementation of BM2 using GuardianMPC, MiniONN [20], TinyLEGO [92], and without any protection and hiding.

Approach	Pre-processing (ms)	Online (ms)		Total (ms)
		Evaluation	Total	
No Protection, no Hiding	N/A	N/A	171.39	171.39
MiniONN [20]	880	N/R	400	1280
TinyLEGO [93]	1961.24	34.76	392.39	2353.63
GuardianMPC	**2323.96**	**2.8**	**425.91**	**2749.87**

TABLE 5: Execution time cost comparison between each phase of implementation of BM3 using GuardianMPC, MiniONN [20], TinyLEGO [92], Chameleon [105], EzPC [106], and without any protection and hiding.

Approach	Pre-processing (s)	Online (s)		Total (s)
		Evaluation	Total	
No Protection, no Hiding	N/A	N/A	9.72	9.72
MiniONN [20]	472	N/R	72	544
EzPC [106]	N/R	N/R	N/R	265.6
Chameleon [105]	22.97	N/R	29.7	52.67
TinyLEGO [93]	913	7.29	154	1067
GuardianMPC	**1191**	**1.18**	**208**	**1399**

is close to MINIONN [20] although GuardianMPC does not have the luxury of SIMD utilized by MINIONN [20].

BM3. Table 5 contains the execution time cost results comparison between the online and offline phase of GuardianMPC, TinyLEGO [92], MINIONN [20], EzPC [106], Chameleon [105], and the BM3 without any protection and hiding. According to Table 5, the online phase execution time cost of BM3 benchmark implementation using TinyLEGO [93] and GuardianMPC is $15.8\times$ and $21.3\times$ than the FPGA-based accelerator implementation of BM3, without any protection and hiding. Moreover, the online phase execution time cost of BM3 is benchmark implementation using TinyLEGO [93] and GuardianMPC is $5.18\times$ and $7\times$ than Chameleon [105], a hybrid secure computation framework.

LeNET-5 Table 6 contains the execution time cost results comparison between the online and offline phase of GuardianMPC, TinyLEGO [92], TinyGarble2 [101], and the LeNET5 [107] without any protection and hiding. According to Table 6, the online phase execution time cost of LeNET5 [107] benchmark implementation using TinyLEGO [93] and GuardianMPC is $18.56\times$ and $21.4\times$ than the FPGA-based accelerator implementation of LeNET5 [107], without any protection and hiding. Moreover, the online phase execution time cost of LeNET5 [107] is benchmark implementation using TinyLEGO [93] and GuardianMPC is $1.83\times$ and $1.46\times$ less than TinyGarble2 [101]. The less online phase execution time cost requirements of TinyLEGO [93] and GuardianMPC compared to TinyGarble2 [101] is due to the memory efficient nature of TinyGarble2 [101] as their main contribution is utilizing less memory peak by sacrificing the performance [101].

TABLE 6: Execution time cost comparison between each phase of implementation of LeNET-5 [107] using GuardianMPC, TinyGarble2 [101], TinyLEGO [92], and without any protection and hiding.

Approach	Pre-processing (s)	Online (s)		Total (s)
		Evaluation	Total	
No Protection, no Hiding	N/A	N/A	1.73	1.73
TinyGrable2 [101]	N/R	N/R	91.1	91.1
TinyLEGO [93]	387.85	2.61	32.1	419.95
GuardianMPC	**429.23**	**0.4**	**37.02**	**466.25**

D. EVALUATION PHASE ACCELERATION ABILITY OF GUARDIANMPC

One of the main contributions of GuardianMPC is to accelerate the evaluation phase of LEGO protocol [34]. This acceleration has been made possible by utilizing the optimization techniques used in HWGN^2 [96] and the parallel processing ability of FPGA.

According to Table 3 and Table 4, GuardianMPC has accelerated the evaluation phase of LEGO protocol [34] $13.44\times$ and $12.41\times$ when executing BM1 and BM2 benchmarks, respectively. The acceleration is also evident in the case of CNNs as GuardianMPC has accelerated the evaluation phase of LEGO protocol [34] by $6.17\times$ and $6.52\times$ when executing BM3 and LeNET5 [107], respectively.

VII. DISCUSSION

A. WHY ACCELERATING EVALUATION MATTERS

Deep learning applications often require real-time or near-real-time processing. In scenarios like autonomous driving, medical diagnosis, or real-time language translation, delays in computation can lead to ineffective or even dangerous outcomes [1]. Minimizing online phase execution time in MPC ensures that the deep learning models can provide timely and relevant outputs [86], [109], [110].

The online phase of the LEGO protocol is divided into three sub-phases [92]: (i) checking, (ii) building, and (iii) evaluation. In this work, we have integrated the HWGN^2 [96] accelerator into the evaluation phase of TinyLEGO [93]. As HWGN^2 [96] has been developed to mitigate semi-honest adversary, it mainly focused on the evaluation acceleration as checking and building phases do not exist in the frameworks flow that are secure against semi-honest adversary model.

It is worth mentioning that the checking and building phases of LEGO [34] can undergo the same acceleration process as the evaluation phase to minimize the online phase of LEGO protocol, as it is the most critical phase in the real-time DL interfaces. Our work is the proof of concepts that it is possible to utilize the optimization method used in $HWGN^2$ [96] and FPGA parallel execution power to decrease the online phase time cost of LEGO protocol. The online phase acceleration are vital in real-time applications such as

B. GURADIANMPC VS. ITS COUNTERPARTS

HE vs. GuradianMPC. HE emerges as a promising solution for oblivious inference, allowing computations on encrypted data. Fully Homomorphic Encryption (FHE) enables this capability, albeit at a higher computational cost.

Partially Homomorphic Encryption (PHE) attempts to reduce this overhead by supporting only a limited set of functions or depth-bounded arithmetic circuits.

Despite the potential, HE-based Deep Learning accelerators face challenges such as the complex implementation of non-linear functions like ReLU, significant computational complexity, and truncation errors [18], [29].

Recent literature [17], [18], [20], [24] discusses the software and hardware implementations of these accelerators. Notably, a hardware accelerator introduced in [111] effectively bridges the gap between FHE and plain computation, leveraging hardware acceleration.

As outlined in [24], both HE and GC, which is the heart of LEGO protocol [34], techniques have inherent limitations. The major limitation of FHE 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, a two-party computation technique, are computationally more efficient as they use symmetric-key cryptographic primitives and benefit from hardware support. However, the primary drawback of LEGO [34], the core of GuradianMPC, is it bears a significant 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.

When selecting a technique for a specific application, it's advisable to consider these trade-offs. While standard HE methods may lack the circuit privacy of Private Function Evaluation (PFE), as offered by GuardianMPC, there are HE protocols that achieve this goal [112], [113].

Zero-knowledge proof vs. GuradianMPC The LEGO protocol [34], as the heart of GuradianMPC, offers a novel approach to secure two-party computation that is notably efficient against active adversaries. Its design utilizes a set of garbled NAND gates which are verified for correctness and then randomly permuted to construct a Yao [114] circuit. This method introduces fault tolerance and significantly reduces the security overhead typically associated with ensuring protection against active adversaries. Especially for larger circuits, LEGO's performance surpasses existing protocols, making it a more practical and efficient solution for real-world applications requiring robust security against active threats.

In contrast, the limitations of Zero-Knowledge Proofs, as discussed in [58], [59], lie predominantly in their practical efficiency. While theoretically robust in offering security guarantees, Zero-Knowledge Proofs tend to be computationally intensive and resource-demanding [59]. This is particularly evident in scenarios requiring security against active adversaries. Their complexity makes them less practical for applications that demand efficient and real-time computations. The LEGO protocol 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 active adversaries [92].

Byzantine-resilient vs. GuradianMPC Byzantine-resilient aggregation [115] is a mitigation technique used in federated learning, a collaborative machine learning [116], [117], to counteract backdoor insertion attacks. It involves applying aggregation rules that are resistant to malicious contributions from malicious parties (often termed as Byzantine clients [115]). The idea is to detect and exclude updates from these clients that could damage or change the overall machine-learning model being trained.

This technique is crucial in a federated learning setup where multiple parties contribute to model training, but ensuring that all participants are trustworthy can be challenging. Moreover, the implementation of Byzantine-resilient aggregation can sometimes lead to a decrease in the overall accuracy of the machine-learning model [116].

GuardianMPC, in contrast to Byzantine-resilient aggregation, takes a different approach by encrypting, garbling, the entire machine-learning process using LEGO protocol [34]. This method ensures robust protection against backdoor insertion attacks without compromising the model's accuracy, as it does not introduce any noise that could alter the model's performance.

Trigger reconstruction vs. GuradianMPC Neural-Cleanse [118] developed the first trigger reconstruction approach [116]. 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.

It may struggle to detect complex or well-disguised triggers that blend in the NN model and can be computationally intensive. Additionally, there's no guarantee of completely removing the backdoor, especially in models with complex features, making it not universally reliable against all backdoor attacks [116].

On the other hand, GuardianMPC, through its implementation of the LEGO protocol [34], offers a robust solution to mitigate complex or well-disguised backdoor triggers with a high degree of probability, something that traditional trigger reconstruction methods may not consistently achieve. Un-

like trigger reconstruction, GuardianMPC's 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 deep learning interfaces where quick response times are crucial. This combination of high effectiveness in detecting advanced threats and efficient computational management makes GuardianMPC a superior choice for DL accelerator protections against backdoor insertion attacks.

Model inspection vs. GuradianMPC 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 deep learning-based anomaly detection and interpretability tools [116]. NeuronInspect [119], DeepInspect [120], and Meta Neural Trojan Detection (MNTD) framework [121] are the cutting-edge approaches that are built based on the model inspection [116].

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 [116]. Additionally, it typically 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 [121], which involves training numerous shadow models. These limitations underscore the difficulties in applying model inspection universally, especially against advanced and evolving backdoor strategies that may not adhere to predictable patterns [116].

In contrast to model inspection, GuardianMPC presents a more universally effective solution against backdoor insertion attacks across all types of neural networks. This effectiveness comes from its core, LEGO protocol [34], independence from the specific functionality and model of the neural network. Additionally, GuardianMPC's strategy of managing computational load between its online and offline phases offers a significant advantage over frameworks like MNTD [121]. This efficient management of computational resources not only enhances performance but also makes GuardianMPC more practical for deployment in real-world scenarios, particularly in DL accelerators. These advantages position GuardianMPC as a superior choice for mitigating backdoor insertion attacks in deep learning accelerators.

Logic Obfuscation vs. GuradianMPC In the context of securing machine learning accelerators, Logic Obfuscation [122], [123] is presented in [124] as a countermeasure against Hardware Trojans. This method involves adding redundant components or locking hardware to ensure it continues functioning even if a Trojan compromises some components or is completely rendered useless without the correct key [124]. However, while effective in certain scenarios, 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 [124].

In contrast, GuardianMPC offers a 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 deep learning models [96] 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 threats such as backdoor attacks by obfuscating the function's core operations, providing a dual layer of security—both in function privacy and in operational integrity; therefore, while logic obfuscation secures hardware by complicating the insertion and operation of Hardware Trojans, GuardianMPC secures both the function and the hardware against a broader range of threats, making it superior in scenarios demanding comprehensive security measures against both internal and external threats.

Formal verification vs. GuradianMPC Formal verification [125], as discussed in [124], 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, where the specialized nature and novel architectures present unique challenges that formal methods still need to extensively address [124].

In contrast, GuardianMPC leverages MPC to provide a more flexible and potentially more robust security layer. This approach does not rely solely on the initial design correctness but continuously protects the data and computation processes even under the operation of potentially compromised hardware. By protecting the computation itself through GC within the MPC framework, GuardianMPC ensures that even if hardware tampering occurs, the integrity and confidentiality of the computation remain intact. Thus, GuardianMPC not only addresses the limitations of formal verification in handling complex and specialized ML accelerators but also provides a dynamic defense mechanism that adapts to evolving threats, making it a superior solution for securing deep learning accelerators against sophisticated hardware-based attacks.

a: *DL backdoor insertion and direct weight manipulation countermeasures qualitative comparison*

We have analyzed the above-mentioned mitigation scenarios for accuracy modification, computational load, scalability, and universality to indicate the advantages of using GuardianMPC as a backdoor insertion and direct weight manipulation attack countermeasures for the DL accelerator.

Table 7 illustrates a qualitative comparison between GuardianMPC and cutting-edge mitigation against direct

TABLE 7: Comparative analysis of various security benchmarks in data processing.

Approach	Maintain Accuracy	Scalability	Real-time Performance	Model Independency
Homomorphic Encryption [17], [18], [20], [24]	N	Y	Y	Y
Zero-knowledge proof [55], [60]	Y	N	N	Y
Byzantine-resilient [115]	N	Y	Y	Y
Trigger reconstruction [118]	Y	N	N	Y
Model inspection [119]–[121]	Y	N	N	N
Logic obfuscation [122], [123]	Y	N	Y	Y
Formal verification [125]	Y	N	N	Y
GuardianMPC	Y	Y	Y	Y

weight manipulation and backdoor insertion attacks. As observable in 7, only GurdianMPC maintains its real-time performance while keeping model accuracy and being scalable and independent from the DNN model. This makes GuardianMPC a better fit for a backdoor-resilient DL accelerator than the cutting-edge mitigation approaches.

VIII. CONCLUSION

In this work, we have introduced GurdianMPC, a backdoor-resilient deep learning accelerator through secure and private function evaluation. By employing Secure/Private function evaluation (S/PFE) within the Multi-Party Computation (MPC) and the LEGO protocol [34], GurdianMPC enhances the resilience of DL accelerators to backdoor insertion threats. The integration of the latest GC optimizations and the computational power of Field Programmable Gate Arrays (FPGAs) has been pivotal in achieving this goal.

Moreover, GurdianMPC not only fortifies DL models against backdoor attacks but also does so with remarkable efficiency. It achieves up to $13.44\times$ less evaluation phase time cost compared to TinyLEGO [93], a cutting-edge framework based on the LEGO protocol. This significant improvement in efficiency is achieved with only a negligible increase in offline computation time cost, marking a substantial advancement in the field.

Last but not least, functional secrecy is a key advantage of GurdianMPC over frameworks like TinyLEGO [93], especially in mitigating sophisticated backdoor attacks in deep learning systems. By leveraging S/PFE, GurdianMPC effectively conceals the internal mechanisms of DL models. This provides resiliency against malicious adversaries and maintains the model's evaluation performance, setting GurdianMPC apart as a secure, efficient, well performed, and resilient solution.

As DL is advancing, the continued exploration and refinement of GurdianMPC will be crucial in addressing the evolving landscape of backdoor insertion threats in DL systems. The integration of advanced cryptographic techniques with cutting-edge computational hardware paves the way for a new era of secure and efficient DL applications. Our work is a step towards realizing the full potential of DL technologies, free from the vulnerabilities that have hindered their progress thus far.

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: June 15, 2023], 2023.
[3] Microsoft Corp., "Azure Machine Learning." [Online] https://azure.microsoft.com/en-us/products/machine-learning/ [Accessed: June 15, 2023], 2023.
[4] "Caffe Model Zoo." [Online] https://github.com/BVLC/caffe/wiki/Model-Zoo [Accessed: June 26, 2023], 2012.
[5] "Keras Pre-trained Models." [Online] https://keras.io/applications/ [Accessed: June 26, 2023], 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] 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.

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

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

[36] Y. Lindell and B. Pinkas, "Secure two-party computation via cut-and-choose oblivious transfer," Journal of cryptology, vol. 25, pp. 680–722, 2012.

[37] 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.

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

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

[40] 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.

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

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

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

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

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

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] 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.

[75] 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.

[76] 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.

[77] 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.

[78] 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.

[79] 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.

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

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

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

[87] 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.

[88] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," in 2017 IEEE International Conference on Computer Design (ICCD), pp. 45–48, IEEE, 2017.

[89] I. Shumailov, Z. Shumaylov, D. Kazhdan, Y. Zhao, N. Papernot, M. A. Erdogdu, and R. J. Anderson, "Manipulating sgd with data ordering attacks," Advances in Neural Information Processing Systems, vol. 34, pp. 18021–18032, 2021.

[90] H. Ma, H. Qiu, Y. Gao, Z. Zhang, A. Abuadbba, A. Fu, S. Al-Sarawi, and D. Abbott, "Quantization backdoors to deep learning models," arXiv preprint arXiv:2108.09187, 2021.

[91] X. Qi, J. Zhu, C. Xie, and Y. Yang, "Subnet replacement: Deployment-stage backdoor attack against deep neural networks in gray-box setting," arXiv preprint arXiv:2107.07240, 2021.

[92] 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.

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

[94] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "Tinygarble: Highly compressed and scalable sequential garbled circuits," in 2015 IEEE Symp. on Security and Privacy, pp. 411–428, IEEE, 2015.

[95] M. Hashemi, S. Roy, F. Ganji, and D. Forte, "Garbled eda: Privacy preserving electronic design automation," in Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, pp. 1–9, 2022.

[96] 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.

[97] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel," in 28th USENIX Security Symp. (USENIX Security 19), pp. 515–532, 2019.

[98] M. Hashemi, D. Mehta, K. Mitard, S. Tajik, and F. Ganji, "Faultygarble: Fault attack on secure multiparty neural network inference," Cryptology ePrint Archive, 2024.

[99] D. M. Mehta, M. Hashemi, D. Forte, S. Tajik, and F. Ganji, "1/0 shades of uc: Photonic side-channel analysis of universal circuits," Cryptology ePrint Archive, 2024.

[100] 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.

[101] 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.

[102] 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.

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

[104] I. Xilinx, "v2021.1." [Online]https://www.xilinx.com/products/design-tools/vivado.html [Accessed: July.3, 2023], 2021.

[105] 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.

[106] 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.

[107] 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.

[108] E. Songhori, H. Siam, and S. Riazi, "Tinygarble framework." [Online] https://github.com/esonghori/TinyGarble [Accessed Jan.30, 2023], 2019.

[109] B. D. Rouhani, S. U. Hussain, K. Lauter, and F. Koushanfar, "Redcrypt: Real-time privacy-preserving deep learning inference in clouds using fpgas," ACM Trans. on Reconfigurable Technology and Systems (TRETS), vol. 11, no. 3, pp. 1–21, 2018.

[110] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "Cryptgpu: Fast privacy-preserving machine learning on the gpu," in 2021 IEEE Symposium on Security and Privacy (SP), pp. 1021–1038, IEEE, 2021.
[111] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in MICRO-54: 54th Annual IEEE/ACM Intrl. Symp. on Microarchitecture, pp. 238–252, 2021.
[112] 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.
[113] C. Gentry, A Fully Homomorphic Encryption Scheme. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.
[114] 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.
[115] 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.
[116] 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.
[117] T. D. Nguyen, P. Rieger, R. De Viti, H. Chen, B. B. Brandenburg, H. Yalame, H. Möllering, H. Fereidooni, S. Marchal, M. Miettinen, et al., "{FLAME}: Taming backdoors in federated learning," in 31st USENIX Security Symposium (USENIX Security 22), pp. 1415–1432, 2022.
[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] X. Huang, M. Alzantot, and M. Srivastava, "Neuroninspect: Detecting backdoors in neural networks via output explanations," arXiv preprint arXiv:1911.07399, 2019.
[120] 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.
[121] 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.
[122] 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.
[123] 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.
[124] 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.
[125] 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.