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

Digital Object Identifier XX.XXXX/ACCESS.2023.DOI

Trusted Deep Neural Execution — A Survey

MOHAMMAD FAKHRUDDIN BABAR, MONOWAR HASAN
Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA
E-mail: {m.babar, monowar.hasan}@wsu.edu

ABSTRACT The growing use of deep neural networks (DNNs) in various applications has raised concerns about the security and privacy of model parameters and runtime execution. To address these concerns, researchers have proposed using *trusted execution environments* (TEEs) to build trustworthy neural network execution. This paper comprehensively surveys the literature on trusted neural networks, viz., answering *how to efficiently execute neural models inside trusted enclaves*. We review the various TEE architectures and techniques employed to achieve secure neural network execution and provide a classification of existing work. Additionally, we discuss the challenges and present a few open issues. We intend that this review will assist researchers and practitioners in understanding the state-of-the-art and identifying research problems.

INDEX TERMS Neural Network, DNN, Trusted Execution, TEE, TrustZone, SGX.

I. INTRODUCTION

In recent years, machine learning architectures have been widely adopted in various domains, including medical and financial applications, sports analysis, telecommunications, cyber-physical systems, and augmented/virtual reality. Many learning models are based on neural network architectures, particularly deep neural network (DNN)-based models. The data used for training models often use large-scale privacy-sensitive information. Hence, failure to maintain the confidentiality and trustworthiness of computation can lead to leaking sensitive information (e.g., health and financial records, personally identifiable information, intellectual property) and have serious consequences. Hence, protecting models and data from adversarial manipulation is crucial to ensure trustworthy learning-based automation. Input data, weight values, and intermediate results determine the output of neural network algorithms. Any manipulation of these parameters may lead to misclassification, as shown in recent attacks [1]–[3]. Further, data used during training can be leaked through the output inference result [4], [5]. Hence, there is a need to design techniques for the trustworthy execution of training/inference tasks that do not leak critical information.

One way to ensure the secure execution of machine learning tasks is to run them inside a "trusted enclave" protected from external adversarial access. Researchers explore various techniques to adapt off-the-shelf trusted computation ideas to enable trustworthy neural network execution. One promising approach is to leverage commodity trusted execution environments (TEEs) [6], [7] such as Intel SGX [8] and ARM TrustZone [9]. However, existing TEE technologies are not designed to execute large-scale, resource-hungry neural network models. Generally, TEEs are equipped with a smaller memory footprint and slower computing units, often insufficient to support neural network training/inference. Retrofitting existing TEEs (or designing a custom one) for secure neural network computation is challenging.

Researchers proposed various techniques to adapt existing TEE technology for executing resource-heavy learning models inside resource-constrained TEEs. While there has been some work understanding solutions in protecting the confidentiality of the learning models and data during computation [10]–[14], there exists no prior work that systematically summarizes *trusted neural network*[1] execution. In this survey, we provide an in-depth review of existing trustworthy neural network execution techniques that use off-the-shelf TEEs (viz., SGX and TrustZone). We systematically *identify the adversarial threats* to enable secure neural network processing. We also provide a *taxonomy* and *classify existing work.*

Methodology & Contributions. We thoroughly studied the last 25 years of literature (1998-2023) and shortlisted 16

[1] We use the term *neural network* and *deep neural network (DNN)* interchangeably in this paper.

papers for this survey that are relevant to our topic. The selected papers cover current state-of-the-art techniques on trusted neural network execution. We crawled the articles related to secure neural network execution in cloud systems, embedded/mobile devices, and edge/IoT applications from the major systems, networking, and security venues. We also manually searched the papers from the websites of the researchers/industry labs that we know of working on similar topics. We primarily use the search keywords "TEE", "Neural Network Security", "Trustworthy DNN", "DNN using TrustZone", "DNN using SGX", among others. We do not include generic machine learning, mobile/cloud/edge computing, and embedded/cyber-physical/control-system papers that do not consider TEE-based security solutions. We exclude TEE security aspects that do not consider trusted neural network integration. We also exclude papers related to general-purpose machine learning security, trusted distributed/federated learning techniques, and those that do not consider neural network architectures.

In summary, this paper makes the following contributions:

- Identification of attack vectors and correlate them with existing mitigation techniques (Section III).
- An in-depth summary and classification (taxonomy) of state-of-the-art trusted neural network execution techniques, in particular those based on SGX and TrustZone technology (Section IV).
- A qualitative comparison among current literature and identification of research gaps (Section V).

We will present the state-of-the-art techniques in later sections but let us now start with the background (Section II) and potential threats to neural network models (Section III).

II. BACKGROUND

We now start with an overview of the TEE (Section II-A) and present background on neural network architectures (Section II-B).

A. TRUSTED EXECUTION ENVIRONMENT

Trusted execution environments (TEEs) [6] complement traditional security measures. TEE is a hardware/software-based security architecture that runs alongside but is isolated from the device's main operating system [7]. TEEs aims to provide an isolated, secure runtime environment to provide confidentiality and integrity for the code and data that can not be exploited even if the host (i.e., main) OS[2] is compromised. This, in turn, excludes the salient components (e.g., OS and security-critical applications) from the trusted computing base (TCB) to reduce the attack surface. TEEs also provide secure storage features where sensitive code and data are stored securely in protected memory blocks. TEEs provide the following functionalities [9]: *(a)* an isolated environment for code execution that cannot be accessed or manipulated by other software running on the same platform; *(b)* a remote attestation mechanism that ensures the integrity of code loaded into TEE; and *(c)* a tamper-resistant design to ensure that data used inside of the TEE cannot be accessed or manipulated externally and that data never leaves the TEE without being encrypted. The major components of a TEE-enabled system are listed below [6].

- *Secure Boot:* A mechanism to ensure that only a piece of trusted code should be loaded to boot the device. Secure boot techniques should prevent starting the system if a modification is detected.
- *Secure Scheduling:* A coordination mechanism between the TEE and the rest of the system to ensure that the processes running in the TEE do not affect the responsiveness and intended behavior of the REE.
- *Inter-environment Communication:* A communication interface that allows the TEE components to interact with the rest of the system. Such protocols must ensure reliability (i.e., memory and temporal isolation), minimum overhead (i.e., unnecessary data copies and context switches), and protection of the communication interfaces (i.e., between REE and TEE).
- *Secure Storage:* A storage space for critical data, that only authorized entities can access. TEEs must ensure the confidentiality, integrity, and freshness of stored data. Designers use secret keys and cryptographic mechanisms (such as authenticated encryption algorithms) to provide secure storage.
- *Trusted I/O Channel:* A secure communication interface between TEE and external peripherals (e.g., keyboard, display, sensors, actuators) so that input and output data are protected from being sniffed or tampered with by malicious applications.

Among various off-the-shelf TEE implementations, Intel SGX [15] and ARM TrustZone [9] are two widely used technologies in many IoT/mobile applications and hence are the focus of this survey. We now present an overview of SGX (Section II-A1) and TrustZone (Section II-A2). We also summarize other proprietary or less commonly used TEEs in Section II-A3.

1) Intel SGX

The Software Guard Extensions (SGX) [15] is a set of security-related instruction codes designed for modern Intel processors. SGX enables both user-level and operating system code to define private regions of memory known as "enclaves." The enclaves are validated by a cryptographic attestation key of the enclave's contents and a hardware root of trust generated by the device vendor. Figure 1 depicts an illustration of SGX architecture. The contents of the enclaves are protected and cannot be read or saved by any process outside of the enclave, even if that external process has more privileges.

SGX operates only on a small amount of data and code, leaving most memory outside the TCB. In SGX,

[2] In literature, the main OS components (i.e., OS kernel, device drivers, libraries, untrusted applications) are often referred to as *rich execution environment (REE)*.

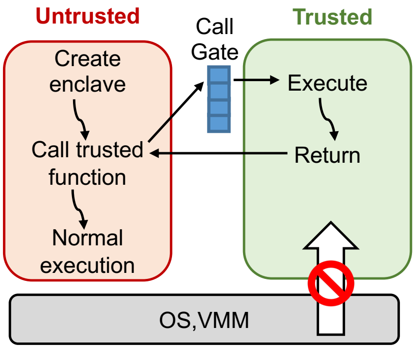

FIGURE 1. Intel SGX Architecture.

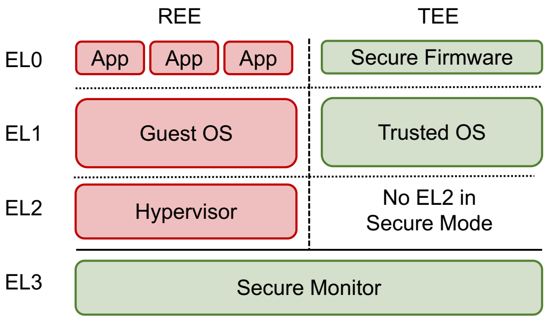

FIGURE 2. Schematic of ARM TrustZone Architecture.

a protected memory region, called "processor reserved memory" (PRM), is safeguarded from any non-enclave memory accesses by the CPU. The PRM stores the "enclave page cache" (EPC), which consists of 4 KB pages containing enclave code and data. The size of PRM is limited to 128 MB, of which 90 MB is reserved for the EPC and the rest for the meta-data required by the "memory encryption engine" [16]. The initial code and data of an enclave are loaded by the untrusted system software (e.g., rich OS). The CPU cryptographically hashes the contents of the enclave, and a remote party can verify the measurement hash through a software attestation process to confirm that it is communicating with a secure enclave. Execution can only enter an enclave through special CPU instructions, and the allocation of EPC pages is delegated to the OS kernel or hypervisor. To prevent private data leakage, the CPU does not directly service interrupts, faults, or VM exits while executing enclave code; instead, it performs an "asynchronous enclave exit" to switch to a predefined area outside the enclave before servicing it. SGX uses cryptographic protections to ensure the confidentiality, integrity, and freshness of evicted EPC pages while in untrusted memory. A detailed description of Intel SGX technology is available in existing literature [8].

2) ARM TrustZone

TrustZone [9] is a hardware/software-based security feature for ARM devices that leverages TEE technology. Figure 2 presents a high-level schematic of the TrustZone architecture.[3] TrustZone divides processor execution states into two main parts: a *"normal world"* and a *"secure world"* (viz., REE). Both normal and secure worlds have their own memory regions (i.e., kernel and user space). The normal world hosts a commodity OS (i.e., acts as REE) and the secure world employs a small, verified, secure kernel. The normal and secure worlds bridge via a software module referred to as *secure monitor*. At any given point in time, a TrustZone-enabled device can execute instructions at four exception levels (EL0-EL3) and two modes (i.e., non-secure mode, secure mode). EL3 (monitor mode) runs the "ARM Trusted Firmware." EL2 is used for the hypervisor, EL1 for the OS kernel, and EL0 is used for executing the application code. The current processor state is determined by a specialized bit, known as a non-secure (NS) bit. The NS bit has two states: NS = 1 for non-secure execution and NS = 0 for secure execution. TrustZone uses a customized mechanism called "secure monitor call" (SMC) for context switching between the two worlds. When an SMC instruction is invoked from the normal world, the processor cores perform a context switch from the normal world to the secure world and freeze their normal world operation. As a security measure, the normal world cannot access secure memory, whereas the secure world can access normal world memory. TrustZone also provisions to isolate the external peripherals. The TrustZone address space controller (TZASC) enforces the physical memory separation. TZASC resides between the system bus and the memory chip. The designers can use TZASC to configure specific memory regions as secure or non-secure.

3) Other TEEs

There also exist other TEE implementations. For instance, the Platform Security Processor [17], (also referred to as AMD Secure Technology) is a TEE subsystem included in AMD microprocessors. Apple uses a separate processor called the SEP (secure enclave processor) [18] for data protection and biometric authentication. Google introduced a similar solution (called Titan M [19]), which is an external security chip that can be found on recent Android Pixel series smartphones. MultiZone Security [20] designed by Hex Five Security [21] is the first TEE for RISC-V architecture.

Among the various TEE implementations, SGX is mostly employed in cloud-based large-scale systems, where TrustZone is widely used in embedded, mobile, and IoT-specific applications. In our study, we were unable to find research articles that use TEEs by other vendors (e.g.,

[3] The TrustZone variant (also called TrustZone-A) presented here is for the ARM Cortex-A family of processors which is widely used in mobile devices. ARM also introduced a TrustZone variant for the Cortex-M family of processors for microcontrollers and low-power IoT devices (called TrustZone-M). Since existing TrustZone-based DNN executions are designed for ARM Cortex-A processors, we exclude the background of TrustZone-M.

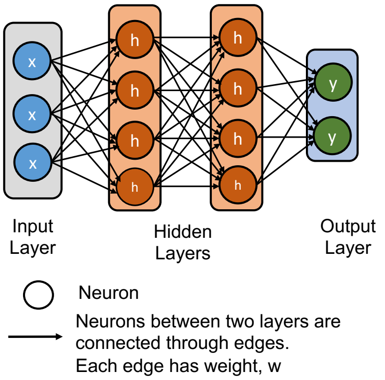

FIGURE 3. A simplified neural network structure. Input and output layers are connected through multiple hidden layers.

AMD, Apple, and Google) or leverage relatively newer technology (e.g., RISC-V MultiZone architecture) for secure DNN processing. Therefore, we limit our focus to TrustZone and SGX-based papers as these two technologies cover the state-of-the-art literature.

B. NEURAL NETWORKS

A neural network, also known as artificial neural network (ANN) [22], is a large-scale, intricate network constituted of layers, including an *input layer*, a couple of *hidden layers* and an *output layer*. Figure 4 depicts a simplified neural network architecture. The naming and structure of ANN are derived from the human brain, and they resemble the way biological neurons communicate with one another. Each layer has many “nerve cells”. The inputs of a current layer’s nerve cells are the outputs of a former layer’s nerve cells. Given training data sets, ANN learns specific parameters of the whole network with *feed-forward* or *feedback*.

A deep neural network (DNN) [23] is a derivative of ANN composed of numerous layers between the input and output. DNNs are typically feedforward networks, where data flows from the input layer to the output layer without looping back. Initially, the DNN constructs a map of virtual neurons and assigns random numerical weights to their connections. The weights and inputs are multiplied, yielding a value between 0 and 1. If the network failed to recognize a specific pattern, an algorithm would adjust the weights. As a result, the algorithm can increase the importance of certain parameters until it determines the correct mathematical manipulation to process the data fully. DNNs consist of four types of layers: *(i)* convolutional layers, *(ii)* activation layers, *(iii)* pooling layers, and *(iv)* fully-connected layers.

Mathematically, a DNN can be represented as a function that maps an input vector $\mathbf{X}$ to an output vector $\mathbf{Y} = F(\mathbf{X})$, where F is the DNN function. Each layer consists of a set of *neurons*. Each neuron takes a weighted sum of its inputs and applies an activation function to produce its output. The output of a neuron can be represented mathematically as $\mathbf{Y} = \sigma(\mathbf{WX} + \mathbf{b})$, where $\mathbf{W}$ is the weight vector, $\mathbf{X}$ is the input vector, $\mathbf{b}$ is the bias term, and σ is the activation function. The purpose of the activation function is to introduce non-linearity into the model, which allows the DNN to learn complex, non-linear relationships between the input and output. The weights and biases in the DNN are learned during the training process. The training process involves iteratively adjusting the weights and biases to minimize a loss function, which measures the difference between the predicted output and the actual output. A commonly used loss function is the *cross-entropy loss* that measures the difference between the predicted probabilities of the different classes and the true probabilities. We can define the cross-entropy loss function as: $L = -\sum y_i * \log(\hat{y}_i)$, where y_i is the true probability of the i-th class, $\hat{y}_i$ is the predicted probability of the i-th class.

There also exist variants of DNNs. For instance, convolutional neural networks (CNNs) [24] are a special case of DNNs that uses matrix multiplications in conjunction with convolutional filter operations and are used for image and video analysis. Recurrent neural networks (RNNs) [25] are specifically designed for time series prediction. RNNs are distinguished by the presence of loops in their layer connections, which allow them to maintain state and make predictions on sequential inputs while maintaining accuracy.

III. ATTACKS ON NEURAL NETWORKS

In most work, researchers assume that the adversary has access to model parameters, input data, and REE subsystem (e.g., rich OS and its memory). Instead of adversary crashing the system, the attacker may secretly infer critical information and leak those for economic gain. The common attack types used in the literature include: *(i)* white-box [26], *(ii)* black-box [27], *(iii)* generative adversarial network (GAN) [28], *(iv)* membership inference [4], and *(v)* fault injection attacks [29], as we present below.

1) White-Box Attack

In a white-box attack [26], the adversary has full knowledge of the model, including its architecture, parameters, gradients and, loss functions. White-box attacks have been extensively studied because the disclosure of model architecture and parameters allows researchers to clearly understand the weaknesses of the models and can be mathematically analyzed. Let us represent a learning model as a function, $\mathbf{Y} = F(\mathbf{X})$, where $\mathbf{X}$ is a feature vector and $\mathbf{Y}$ is an output vector. In a white-box attack, attackers attempt to generate the desired adversarial output $\mathbf{Y}^*$ by constructing an adversarial sample $\mathbf{X}^*$ from a sample $\mathbf{X}$ by adding a perturbation vector $\delta_{\mathbf{X}}$.

2) Black-Box Attack

Unlike white-box attacks, in black-box attacks [27] the adversary can only get outputs for provided inputs and has no knowledge of the model structure or parameters. Hence, the inner configuration of DNN models is unavailable to the adversary in a black-box attack scenario. In a black box attack, the adversary attempt to build their own model F^*. Then the attackers generate an adversarial sample $\mathbf{X}^*$ and apply the adversarial sample to the target model F.

3) Generative Adversarial Network (GAN) Attack

GAN [28] is an unsupervised learning model where two neural networks compete to become more accurate in their predictions. Generative modeling includes automatically detecting and learning the patterns from the input data. However, the goal is to train in a way that the model can be used to generate new samples that seem to draw from the original dataset. The two neural networks that comprise a GAN are known as the generator model G and the discriminator model D. The generator model G learns the training data distribution by attempting to generate "fake" samples that are difficult for the discriminator D to distinguish from the real ones. The generative network's main goal is to "fool" the discriminator network by producing new candidates that the discriminator thinks are not synthesized but are part of the true data distribution. Simultaneously, the discriminator model D learns to differentiate whether a given sample comes from the training data or is created by the generator G. The training objective of G is to increase the error rate of the discriminator D.

4) Membership Inference Attack (MIA)

MIAs [4] are a type of attack that uses "memorized information" from the layers of a model to identify whether a given data record was part of the training dataset. MIAs could be performed both in black-box and white-box settings. In a black-box MIA, the attacker uses model outputs and auxiliary information (such as public datasets or the public prediction accuracy of the model) to train shadow models or classifiers without gaining access to the internal model parameters. For instance, to obtain the outputs of the model F, the attacker trains a binary classifier C with labeled data points to predict whether a given data point was in the training dataset. Based on the model's output, the classifier C assists the attacker in inferring data points for the training set.

In a white-box MIA, the attacker has the internal information of the model architecture. For instance, the learning model $\mathbf{Y} = F(\mathbf{X})$, where $\mathbf{X}$ is a feature vector and $\mathbf{Y}$ is a prediction, is known to the attacker. Adversaries can thus create true samples and use active learning techniques on these known samples to create a very accurate dataset that closely resembles the original dataset.

5) Fault Injection Attack

A fault injection attack [29] may involve modifying the network's input data or internal parameters that lead to incorrect or unexpected outputs. An attacker could, for instance, introduce noise into the input to cause the DNN to misclassify. Mathematically we can represent fault injection attack as a conditional probability distribution $P(\mathbf{Y}|\mathbf{X})$, where $\mathbf{Y}$ is the compromised output and $\mathbf{X}$ is the expected output (i.e., without any attack). The difference between the probability distributions of $\mathbf{X}$ and $\mathbf{Y}$ is used to determine the success of such an attack.

A. THREAT MODELS USED IN LITERATURE

The majority of the work (i.e., 11 out of 16) [30]–[40] assume black-box attacks and the rest consider white-box attacks [41], GAN-based attacks [42], fault injection attacks [43], [44], and membership inference scenarios [45]. A typical assumption is that adversaries have complete access to the rich OS. For instance, an attacker may compromise the integrity and confidentiality of the target DNN model, including modifying the input, reconstructing the model, and uncovering the final inference result of the model (HybridTEE [30], Slalom [35], Sectum [39]). A stealthy attacker could try to access the model's parameters, intermediate results, and final output without disrupting the inference task (T-Slices [40]). Further, the adversary may compromise the rich OS to intrude, forge, and modify the interference task and steal user data from non-protected processes (SecDeep [31], real-time DNN [38]). They may also have access to the cache, memory, and storage (Trusted-DNN [33]) and inject *faults* that may change specific DNN parameters in the memory (AegisDNN [43]). Using such access, the adversaries could *(a)* steal input and output data (e.g., users' private information) and model parameters (e.g., pre-trained weight data) and *(b)* change data and compute false results to mislead users. It could also be possible that the adversary can steal user input but provide correct output — say to glean private user information managed by a malicious party but remain stealthy (Serdab [32]). One example could be home automation systems, where home camera feeds are stored and processed using deep learning models by a third-party cloud vendor. Hence, a trusted DNN execution framework can protect the privacy of user inputs (e.g., camera feeds) hosted by the (potentially malicious) edge/cloud providers. Consider a case an untrusted user has complete control over the OS, memory, and storage. The attackers then may attempt to obtain proprietary information (e.g., device configuration and other intellectual property) such as those available unencrypted in memory or storage but not publicly accessible (Confidential-DL [34]). The attacker may have access to a remote node where the DNN models/data are accessible once they are transferred from a cloud server (DarkneTZ [41]). The adversary may seek to extract sensitive data, such as vendor intellectual property, user input/output, or data that identifies or tracks the user (OMG [36]).

In GAN-based attack scenarios, all intermediate model data could be accessible to adversaries after offloading the second partition of the DNN in an intermediate layer to an unsecured CPU or GPU (Origami [42]). The goal of the attacker is to reconstruct the input using GAN. For multi-party scenarios (e.g., user, developer, and a cloud

TABLE 1. Summary of Adversarial Capabilities

Reference	Attacker's Access/Capabilities	Attack Type
AegisDNN [43]	Inject faults data	Fault injection
Confidential DL [34]	Unencrypted data of the model	Black-box
DarkneTZ [41]	Access to some part of DNN models	White-box
HybridTEE [30]	Modify the input, reconstruct model, extract the final inference results	Black-box
Occlumency [37]	Access to cloud software hardware resources	Black-box
OMG [36]	Vendor intellectual property, User input and output data	Black-box
Origami [42]	Access to some part of DNN models	GAN
Privado [44]	Access to the DRAM	Fault injection
Real-Time DNN [38]	Access to input data, known task periods and execution times	Black-box
Sclera [45]	Training data	MIA
SecDeep [31]	User data of non-protected side	Black-box
Sectum [39]	Input Data, Model information	Black-box
Serdeb [32]	User input	Black-box
Slalom [35]	Information about model or inputs	Black-box
Trusted-DNN [33]	Device cache, memory, and storage	Black-box
T-Slices [40]	Access to the model parameters and intermediate results	Black-box

vendor), a rouge vendor may try to steal user inputs or alter inference models (Occlumency [37]). The cloud provider may leveraging software vulnerabilities to obtain privileged system access and acquire sensitive data or capture information flow by exploiting existing side-channel attacks (Privado [44]). Further, a truthful cloud provider may obey the protocol but stealthily intend to steal client's private information by launching a membership inference attack (Sclera [45]).

Table 1 summarizes threat models and assumptions on the adversarial capabilities used in the literature. As we can see from the table, a common assumption is that the adversary has access to model parameters, which are stored on the untrusted side. They may also be able to change model parameters. However, they do not have access to the data of the trusted OS — this is a reasonable assumption as TEEs are intended to protect critical information. While there exist some side-channel, physical, and DoS attacks on TEEs [46]–[53], the state-of-the-art TEE-based DNN execution models do not consider such attack vectors.

IV. TRUSTED DEEP NEURAL EXECUTION TECHNIQUES

Many DNN-based applications (such as image processing, object detection, medical records, and financial transactions) deal with sensitive data and require protection against tampering or theft. While TEEs could be a promising approach to protect critical DNN applications, a TEE-based architecture must adhere to limited trusted resources available. Many widely-used DNN models deal with a large number of parameters. For instance, LetNet [54], AlexNet [55], and VGG16 [56] have 60 thousand, 60 million, and 138 million parameters, respectively [57]. The execution of such models inside TEEs needs a significant amount of storage and memory. Secure enclaves often have small memory. For instance, TrustZone has only 8 MB of memory [58] and SGX has up to 128 MB of secure memory [15]. Unlike traditional systems, hence it is more challenging to run DNN models in a TEE-based architecture.

TABLE 2. Trusted Neural Network Execution Techniques

Reference	Key Idea
AegisDNN [43]	Sensitive layer in TEE based on a given protection configuration profile
Confidential DL [34]	Layer-by-layer execution inside the TEE
DarkneTZ [41]	First l layers in the REE and the remaining layers inside a TEE
HybridTEE [30]	First few layer in a local TEE (TrustZone), intermediate layers in cloud TEE (SGX), and the final layer in the local TEE (TrustZone)
Occlumency [37]	Offloading user data to SGX enclave (cloud server) and execute the model inside the enclave
OMG [36]	Encrypted data exchange between user and vendor such that both party cannot infer secret information
Origami [42]	Similar to Slalom, but use blinding and deblinding factor only for the first few layers
Privado [44]	Generates enclave-specific code and encrypted parameters of the models that is loaded to the server to prevent leakage
Real-time DNN [38]	Fused various layer from same or multiple task subject to TEE capacity
Sclera [45]	Splitting the model into multiple sub-model based on privacy expectation & resource availability
SecDeep [31]	Sensitive model code and data inside the TEE
Sectum [39]	Offload deep learning tasks to the edge-cloud running trusted enclave
Serdeb [32]	Plaintext tensor-like matrix multiplication inside the TEE
Slalom [35]	Calculates matrix multiplication in a faster device using blinding and deblinding technique
Trusted-DNN [33]	Partition the encrypted weight file into variable-length size to execute all layers within the TEE
T-Slices [40]	Converts deep learning models into slices and executes as many slices as possible sequentially inside the TEE

Researchers propose various techniques to execute DNN models inside resource-constrained trusted enclaves. This section presents various ideas available in the literature. Table 2 summarizes the key approaches used by the authors. We divide the TEE-based execution techniques into two major classes: *(a)* full execution (all DNN layers running

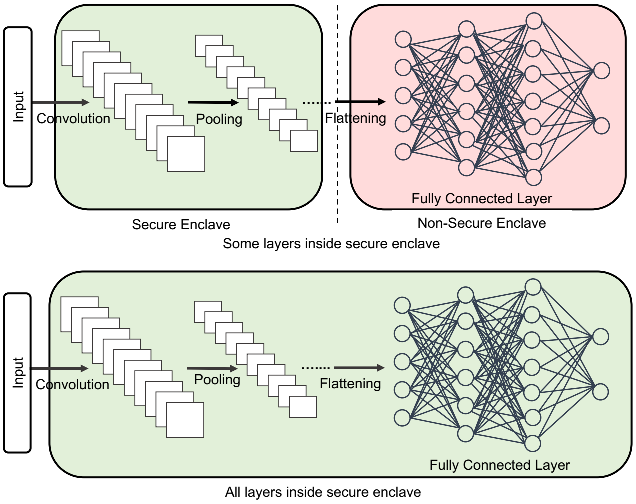

FIGURE 4. High-level illustration of trusted DNN execution approaches. The trusted enclaves (e.g., SGX/TrustZone) can be used to execute only the critical layers (top figure) or all of the layers (bottom figure), depending on the design configuration.

inside TEE) and *(b)* partial execution (only a subset of layers running inside TEE). The bottom part of Fig. 4 depicts a case where all layers run inside the enclave where the top parts illustrate a partial execution scenario. We discuss several full and partial execution techniques in Section IV-A and Section IV-B, respectively. Section IV-C presents various solutions designed for targeted use cases (e.g., cloud and real-time computing, computation offloading, and privacy-aware executions).

Figure 5 presents a classification of existing work. We categorize the papers based on the number of TEE technology (i.e., single or multiple TEE architecture). Each of them may support single or multiple enclaves. We further classify the techniques based on whether all or some parts of DNN layers run inside the TEE (e.g., SGX, TrustZone, or a combination of both). We present the summary of our findings in Section V-A.

A. COMPLETE EXECUTION: ALL LAYERS RUNNING INSIDE TEE

One way to enable trusted DNN execution is to run the complete DNN application inside the TEE. However, as mentioned earlier, TEEs have resource constraints and may not fit a large DNN model. Hence researchers propose various techniques to “trim” the model and fit it within the enclave capacity, as we present below.

Peter et al. [34] propose a TrustZone-based confidential deep learning framework by breaking down the model layer-by-layer to tackle the TEE’s memory constraints. The key idea is to use “layer-based partitioning” where each layer is an independent partition. Each partition includes its weights and biases and is stored in a separate encrypted file. An encrypted file of a partition is loaded into shared memory and decrypted by a trusted application on the secure side. Then the necessary computation is done to activate all the neurons belonging to that partition. After computation, the current layer’s weights and biases are discarded. The activation information of the neurons is then stored in a secured memory and used as input to the next layer. The partitions may still need a large memory if a single layer has many neurons. The authors introduce a “sub-layer partitioning” approach to solve this issue. In this method, the model is partitioned by a layer as before. Then, each layer is further divided into p subsets to address the memory constraints. Sub-layer partitioning and layer-based partitioning have some commonalities for the following steps: *(i)* sending the encrypted data (activation, weights, bias) of the layer, *(ii)* calculating the activation of the present layer, *(ii)* discarding the weights and bias of the previous layer, and *(iv)* using the stored activation data for the incoming layer.

Trusted-DNN [33] is a memory-aware, encryption-based model protection strategy based on TrustZone. This technique ensures the privacy and integrity of user input data, model weights, and output data by encrypting input/output data with the RSA encryption algorithm. The *client* and the *trusted application* produce RSA keys and send the public keys to each other before starting the DNN execution. Trusted-DNN leverages TrustZone’s safe storing technique to protect the weight data. To handle cryptographic keys, TrustZone uses a multi-level key management unit based on a hardware unique key (HUK) [59] that is only read by TEE. The TEE uses HUK to generate a trusted applicant storage key (TSK) [59] for each TA. The DNN model’s weight file is encrypted with a randomly generated AES [60] key, which is then encrypted with TSK. This ensures that only a specific TA can read the file. Trusted-DNN divides the encrypted weight file into parts of varying sizes, which can be changed to fit the memory before each execution. Only one segment is loaded into a memory buffer of the same size at any given point in time. The buffer’s content changes when weight parameters from another segment are needed during the computation. Trusted-DNN further supports a special convolutional layer calculation approach [33] that allows finishing an inference by traversing the weight file only once from front to back. Instead of computing with a convolution kernel, Trusted-DNN calculates all results linked to one parameter independently in a convolution kernel. Then, it accumulates the results individually to the relevant place in the output feature map. Trusted-DNN also employs a model compression approach (i.e., binary quantification compression) that reduces the number of weights and computations in a DNN model, thus making it suitable for TEE-enabled systems.

Note that partitioning the DNN model and then running all the layers of those partitions can still take a large amount of time. Generally, the DNN partitions contain two parts: *(a)* a small secure component that runs in the protected TEE, and *(b)* a large non-secure part that runs outside of the TEE.

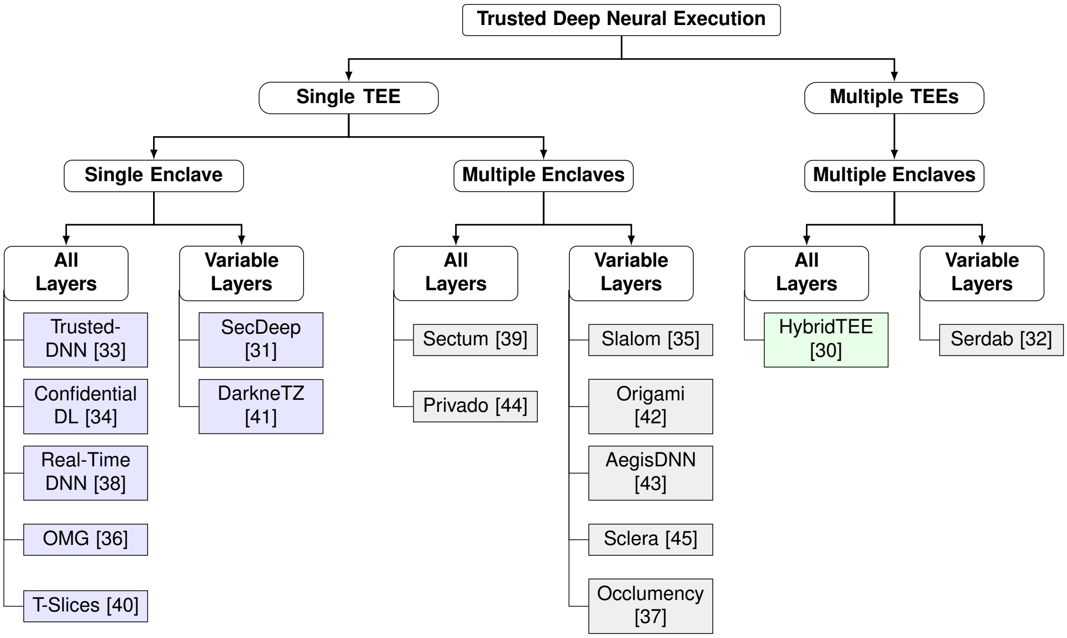

FIGURE 5. A classification of existing Trusted DNN frameworks. The system may support a single TEE architecture or a combination of a few (say both SGX and TrustZone). Each TEE architecture may consist of single or multiple enclaves where some or all part of the DNN layers are executed. We use Blue and Gray nodes to represent TrustZone and SGX-based designs, respectively and Green node depicts a combination of both TEE technology.

The non-secure part could pose a threat to user privacy as it can be accessible by the adversary. An attacker may get sensitive information from the non-secure part, that in turn, can aid to compromise the DNN model. HybridTEE [30] aims to trade off security and execution time and ensure user privacy. The goal is to ensure the safety of the data and the performance of the DNN model execution. HybridTEE achieves this by running the entire DNN model in a local secure environment (TrustZone enclave) and offloading the DNN model on the SGX server. HybridTEE introduced a trustworthy communication channel connecting TrustZone and SGX to enable secure offloading of the DNN between two TEEs. Running the whole DNN in the TEE, HybridTEE ensures the security and confidentiality of the data. HybridTEE offloads the DNN model to a remote Intel SGX server to achieve the desired performance. The DNN model is partitioned into LocalNet, RemotNet and PredNet. LocalNet contains the most sensitive part of the DNN. The data and code of the DNN and its feed-forward function should reside in the local TEE (i.e., TrustZone) for privacy protection. RemotNet contains the greatest number of layers as it runs in the high-performance SGX server.

In HybridTEE, an auxiliary DNN is used for object detection at the intermediate layers. If the auxiliary DNN is not able to classify an image with a certain degree of confidence, the image is then considered free of privacy exposure and can be offloaded to a remote server. Finally, PredNet — the final layer of the DNN layer — which performs the final interference, executes inside the local TEE (i.e., TrustZone) as the inference result is more sensitive.

Occlumency [37] presents a cloud-based solution that protects user privacy without limiting the benefits of utilizing powerful cloud resources. Occlumency uses a secure SGX enclave to maintain user data confidentiality and integrity throughout the DNN inference process. Since on-device inference leads to high energy costs in battery-operated smart devices, Occlumency addresses this problem using a cloud-driven approach. Through a secure channel, a device sends user data directly into an SGX enclave on a cloud server. Occlumency then runs the entire DNN processing pipeline in the enclave and returns the results to the device. As a result, Occlumency secures users' private input data, inference results, and all intermediate outputs throughout the offloading process while leveraging powerful cloud resources. However, due to limited physical memory space and inefficient page swapping, DNN inference in the SGX enclave imposes a significant performance degradation [61]. Occlumency addresses this by using intelligent techniques such as *(a) on-demand weights loading* that dynamically load a part of model weights in the protected enclave, as needed, *(b) memory-efficient inference* that reduces memory usage to store the intermediate data needed for inference and *(c) parallel pipeline* for weights copying, hash checking, and model inference to best utilize the SGX resources.

Researchers also proposed techniques that do not sacrifice accuracy while ensuring trustworthy inference. For instance, T-Slices [40] proposes a new inference framework that dynamically converts an unmodified deep model into smaller units (called *slices*). Each slice executes independently and remain within the trusted memory limit. T-Slices sequentially loads as many slices as possible into the trusted memory, while the next slices wait in the encrypted untrusted memory. The prediction accuracy is not impacted since T-Slices does not modify the protected model.

B. PARTIAL EXECUTION: VARIABLES LAYERS RUNNING INSIDE TEE

Running all DNN layers inside the enclave incurs significant timing overhead. An alternative option for faster inference could be executing only a few layers inside TEE and running the rest in CPU/GPU. The challenge then is to determine *which layers* should be executed inside TEE. We now discuss DNN execution models that run a variable number of layers inside TEE.

DarkneTZ [41] presents a partitioning algorithm — it divides the DNN model into two parts: *(a)* a non-sensitive layer and *(b)* a sensitive layer. The non-sensitive layers are executed in REE. In contrast, the sensitive layers are executed in TEE which allows the computation of the whole DNN model on devices with limited resources. Splitting models into two segments can avoid training data from being leaked using inference attacks (e.g., MIA). For instance, if there exist L layers in a DNN model, the model owner determines an intermediate layer l where $1 \leq l < L$. During the fine-tuning or inference of DNN, the (untrusted) client application runs layers 1 to l in the REE while the (trusted) server application runs the remaining layers inside the TEE (i.e., layer $l+1$ to L). The rationale for running the last few layers in TEE is that those layers have a high probability of leaking information about the training data. If there exists an encrypted layer (say, given by the model owner), that layer will be decrypted by the secret key stored inside the TEE. We note that running a part of the model outside the enclave reduces verifiability and increases latency.

SecDeep [31] uses an on-device accelerator (ARM Mali GPU) which provides two orders of magnitude better inference performance than the on-device processor (ARM Cortex CPU). To figure out which parts of the inference code and data will run inside the TEE, SecDeep divides deep learning libraries into two parts: *(a)* the confidential computing base, which runs in the TEE, and *(b)* the non-confidential computing base, which runs in the untrusted execution environment. At development time, the confidential and non-confidential computation bases are unidentified preprocessor directives so that they can be separated at compile time. SecDeep starts with a secure boot process and works for each layer in an iterative manner. For instance, when any data needs to be sent to the non-confidential computing base, it first encrypts data inside TEE. Then, the non-confidential computing base uses the encrypted data and the model parameters and applies the required neural processing for the current layer, and sends the results to the confidential computing base. The above process is repeated for the remaining layers.

Elgamal et al. [32] propose a distributed framework (named Serdab) for deploying DNN across multiple secure enclaves (e.g., SGX) that guarantees privacy and confidentiality. The key idea is to distribute neural network layers across multiple enclaves when running the entire DNN framework in a resource-constrained TEE is inefficient. A “placement path” mechanism determines whether a layer will be executed on a trusted resource or an untrusted resource. In Serdab, the first layer must run on a trusted enclave. If a layer is to be executed on an untrusted device, the input to that layer should be significantly dissimilar to the input of the first layer. The similarity is defined by the similarity function (i.e., correlation, image resolution) and a threshold. For example, let us consider Layer 4 of a neural network is significantly less similar to the original image. Then, the first four layers will be executed in TEE, and the remaining layers can be executed on a regular processor (e.g., hardware accelerators). Serdab can speed up inference through parallel processing. For example, when one TEE is processing some part of the neural chain for a given video frame, another TEE can start processing the initial part of the neurons for the following frame.

C. APPLICATION-ORIENTED SOLUTIONS

So far we discuss approaches that develop partitioning techniques for trustworthy DNN execution. Researchers also investigate various ideas geared toward targeted use-case such as securing matrix multiplications in a DNN model, providing DNN as a cloud/edge service, retaining model/data privacy for multi-party communication, and adapting TEE-based DNNs for latency-critical (viz., real-time) applications, as we discuss below.

1) Outsourcing Matrix Multiplication

Matrix multiplications account for a significant portion of the arithmetic operations required to evaluate DNNs, in both convolutional and fully connected layers. Slalom [35] aims to take advantage of faster devices to calculate the matrix multiplication in order to speed up the secure DNN execution process. In particular, Slalom proposed to partition neural network computation into a forward phase executed in an untrusted environment and a backward phase in a trusted environment. Therefore, it delegates the execution of all linear layers (e.g., matrix multiplication, addition, and other operations, that account for 99% of the DNN computation) from a TEE to a faster (though untrusted) processor. The matrix multiplication outputs are then sent back to the secure enclave and verified using Freivalds' algorithm [62].

Origami [42] allows privacy-preserving inference for massive DNN models by leveraging *cryptographic blindings* (i.e., obscuring data by introducing noise). Origami divides the DNN model into several partitions. The first partition (running inside the SGX enclave) receives encrypted user input. After decrypting the input, the input data and model parameters are cryptographically blinded. Origami then sends the obfuscated data to an untrusted GPU or CPU for processing. When the computation is offloaded to a GPU or CPU, the SGX keeps the blinding and de-blinding factors private, preventing the data from being inferred from the adversary through reverse engineering. Similar to Slalom [35], the outsourced matrix multiplication process is repeated for each DNN layer. The overhead of blinding and unblinding the data is a limiting factor to scalability. The authors show that Origami results in 11x to 15x improved inference performance over Slalom.

2) Trusted DNN on the Cloud & Edge — Learning-as-a-Service

Situations where an enterprise that does not have an in-house infrastructure for large-scale DNN processing may need to rely on third-party cloud services for such computation. PRIVADO [44] aims to aid such scenarios by providing secure DNN *inference-as-a-service*. PRIVADO makes C/C++ based DNN frameworks “input-oblivious” and removes input-dependent access patterns to prevent information leakage. In PRIVADO, an automated tool (called PRIVADO-Converter) detects data-dependent access patterns and uses existing oblivious constructs methods [63]–[65] to modify them in order to make the model data independent.

SCLERA [45] also provides learning-as-a-service and supports secure execution of client workloads using SGX enclaves. To reduce the computational complexity and achieve low-latency inference, SCLERA leverages DNN splitting and quantization, enclave parallelization, and a resource-aware offloading policy that protects clients’ private data.

3) Privacy-preserving Enclaves

Consider an interaction between a user U and vendor V providing learning services. U provides input data (say, voice transcripts) to V and is concerned about data privacy. V shares machine learning models that contain intellectual property. Hence U must not be able to reverse engineer the model. OMG (Offline Model Guard) [36] can assist such scenarios by keeping information about the model’s architecture or user data secret. OMG relies on a small TCB TrustZone extension (called SANCTUARY [66]) and allows the secure execution of machine learning models on mobile devices. OMG comprises three phases: preparation, initialization, and operation. The enclave is loaded and attested by both U and V. Then, V provides an encrypted model to the enclave. During initialization, V passes the model’s decryption key to the enclave. Finally, the enclave is ready for processing and the user U sends input data for running learning-related tasks.

4) Time-aware Trusted DNN Execution

Many safety and latency-critical systems (such as autonomous vehicles and manufacturing robots) DNN models need to perform correctly within a stringent timing budget. Malicious actions such as fault injection attacks may lead to delays in the processing and result in erroneous outputs, thus threatening the safety of the system. AegisDNN [43] aims to protect critical parts of the time-sensitive DNN tasks from fault injection attacks by running them inside SGX enclaves. AegisDNN measures the amount of misclassification that can occur when a DNN layer is not properly protected. AegisDNN uses this information to determine the layer protection configurations for a given task set in order to meet both temporal guarantees and dependability requirements.

A similar line of work exists [38] that presents a technique to enable time-critical DNN computations for TrustZone-enabled enclaves. As TrustZone enclaves do not have enough capacity to process DNN models from multiple tasks and run them together while retaining temporal constraints, the authors proposed splitting the DNN tasks into multiple chunks. As a result, the proposed technique can fit larger models inside a resource-constrained enclave and reduce the context switch overheads and making time-sensitive DNN computation plausible.

Sectum [39] performs an empirical study to analyze the relationship between DNN inference latency and memory occupation. The authors design a latency predictor for SGX-hosted inference, which predicts inference latency using a two-stage approach. The first stage uses a graph neural network (GNN)-based model to detect whether a given model would trigger memory over usage. The second stage combines operator-level (e.g., ADD, MUL, CONV_2D) profiling with linear regression to predict the latency of a model.

V. DISCUSSION

We now summarize our findings (Section V-A) and outline a few open research issues (Section V-B).

A. SUMMARY AND OBSERVATIONS

Efficient neural network training and inference while maintaining privacy, integrity, and confidentiality is a challenging task. Researchers proposed several strategies depending on application requirements. For instance, there exist models for running the entire DNN layers inside TEEs [30], [33], [34], [38], [40]. Another alternative is to run the model in multiple enclaves to speed up the inference process [30], [32], [35], [37], [39], [42], [43], [45]. We can also run only the sensitive layer(s) inside TEE and the rest in an untrusted environment (e.g., REE) [31], [32], [41], [43]. Table 3 summarizes the techniques we discuss in this survey. Majority of work (14 out of 16) [30]–[38], [40], [42]–[45] focus on inference task and remaining two [39], [41] consider both training and inference (see “Learning Task” column). We use the “Design Focus” column to demonstrate the key performance metrics considered by the

TABLE 3. Qualitative Comparisons of the Papers Studied in this Survey

Reference	Target Application	TEE Technology	Enclave	Learning Task	Design Focus
AegisDNN [43]	Cloud Systems	SGX	Multiple Enclaves	Inference	Privacy, Performance
Confidential DL [34]	Platform/application oblivious	TrustZone	Single Enclave	Inference	Privacy, Confidentiality
DarkneTZ [41]	Smartphone, IoT	TrustZone	Single Enclave	Training and Inference	Privacy
HybridTEE [30]	Mobile Devices	TrustZone + SGX	Multiple Enclaves	Inference	Privacy, Performance
Occlumency [37]	Mobile Devices and IoT	SGX	Multiple Enclaves	Inference	Privacy, Performance
OMG [36]	Smartphone, Tablet	TrustZone	Multiple Enclaves	Inference	Privacy
Origami [42]	Cloud Systems	SGX	Multiple Enclaves	Inference	Privacy, Performance, Low Overhead
Privado [44]	Cloud Systems	SGX	Multiple Enclaves	Inference	Privacy, Performance
Real-time DNN [38]	Embedded Devices	TrustZone	Single Enclave	Inference	Performance, Low Overhead
Sclera [45]	Cloud Systems	SGX	Multiple Enclaves	Inference	Privacy, Performance
SecDeep [31]	Mobile Devices and IoT	TrustZone	Single Enclave	Inference	Privacy, Confidentiality
Sectum [39]	Cloud Systems	SGX	Multiple Enclaves	Training and Inference	Privacy, Performance
Serdab [32]	IoT	SGX	Multiple Enclaves	Inference	Privacy, Performance
Slalom [35]	Cloud Systems	SGX	Multiple Enclaves	Inference	Privacy, Performance
Trusted-DNN [33]	Embedded Devices	TrustZone	Single Enclave	Inference	Privacy, Low Overhead
T-Slices [40]	Embedded Devices	TrustZone	Single Enclave	Inference	Privacy, Low Overhead

authors. We classify them into four major categories: *(a) privacy* (user data and model secrecy), *(b) integrity* (keeping data/parameters from being tampered), *(c) performance* (faster runtime, say by offloading to some layers on faster processing units such as GPU or cloud), and *(d) low overhead* (reduced TEE memory footprint, context switches, and cryptographic computation).

We find that there is a trade-off between security and performance. For instance, for faster processing, we may need to run the (non-sensitive) layers on powerful (and potentially untrusted) components (e.g., in GPU). Running all layers inside TEE incurs a significant timing overhead. Further, running the entire model insider enclaves may often be infeasible as most TEEs have limited, finite memory. Partitioning the model and executing the only sensitive layers within a TEE could be an alternative to address this problem. For example, in image/voice recognition applications where the user may not want to reveal input and processed data, running initial input layers and final output layers inside TEE should be sufficient.

B. OPEN RESEARCH ISSUES

While there exists some work on trusted neural network computation, research in this domain is still in the early stages. In the following, we identify a few research challenges and open issues.

1) Resource-aware Trusted Learning

Each inference in a learning model necessitates a large amount of computing, which consumes a significant amount of energy. Energy consumption is critical on the low-resource edge and IoT devices [67], [68]. Further, they often have limited computational resources (e.g., processor, memory, and storage). Data and DNN parameters are usually stored in memory, where significant memory is consumed by frequent memory access [69]. Transferring data from a low-cost external memory to an expensive on-chip memory leads to a considerable increase in energy consumption. Neural network executions incur significant power, memory, and timing overhead. Hence, developing energy-aware secure DNN execution in edge devices is critical. One of the research challenges is to address the issue of frequent memory access and design an efficient memory hierarchy architecture and data-flow mapping strategy. Further latency-critical applications, such as future autonomous vehicles need to perform online training and inference within a stringent timing budget. We find that only limited studies exist [38], [43] for enabling trusted learning. We may need to explore the inner dynamics of the models and identify their less influential weights to make them lightweight without sacrificing accuracy. Developing a holistic resource-aware TEE-enabled learning framework requires further research.

2) Attack-resilient Frameworks

Existing trusted learning frameworks assume that underlying TEE technology is secure. However, TEEs are often prone to side-channel attacks that may leak critical information such as execution time, thermal usage, and memory access patterns, among others [70], [71]. For instance, researchers demonstrated a side-channel against DNNs on SGX-enabled cloud servers that can infer encrypted inputs from the model [44]. There have been limited studies that explore side-channel vulnerabilities in the context of TEE-enabled secure neural network execution, in particular for embedded and IoT-style systems. We believe a concerted effort is required to develop an end-to-end secure learning framework that protects critical information against multiple classes of attacks including side-channel, MIA, and white-box attacks.

3) Exploration of Other TEE Architectures

Our study shows that most frameworks consider SGX and TrustZone (i.e., x86 and ARM architecture, respectively) as underlying TEE technology since they are the predominant TEE platforms as of today. Other architectures such as RISC-V [72] getting popularity to cope with the performance and power requirements of mightier workloads required by emerging AI and learning-driven applications. We believe that the exploration of enabling trusted learning for other TEE architectures, especially RISC-V MultiZone [20] is a promising area of research.

VI. RELATED SURVEYS

There have been limited studies that survey trusted neural network computation. Perhaps the closest line of work is Duy et al. [14] which survey various machine learning frameworks in the context of "confidential computing." The authors summarize attack vectors and review existing techniques. Unlike ours, the focus is on generic learning models and does not provide an in-depth analysis of neural network models. On a related study, Sagar et al. [13] classify confidential learning techniques based on cryptographic primitives and hardware-based protocols. However, the authors' study is limited to SGX-based TEEs only. A similar systematization is performed by Mo et al. [73]. In this (non-peer-reviewed) technical report, the authors discuss how the notion of confidential computing can be used to increase security and privacy in various machine learning settings. In contrast, our study primarily reviews TEE-enabled neural network execution frameworks and presents a taxonomy and classification of existing literature. Besides, the above work do not cover recent papers (2021 and beyond). Our study complements existing confidential computing-based learning surveys. There exist other surveys that review *(a)* generic (viz., insecure/untrusted) DNN computation for IoT-specific devices [74], mobile applications [75], *(b)* adversarial threats to machine learning [10], and *(c)* hardware security for DNN models [76], among others. However, the consideration of TEE-enabled neural network models distinguishes our survey from existing reviews.

VII. CONCLUSION

This paper thoroughly reviews state-of-the-art techniques for enabling commodity "trusted environments" for neural network training and inference. To the best of our knowledge, this work is one of the first efforts to systematically summarize existing literature and threat vectors in the context of trusted and privacy-preserving neural network computation. Our threat classification, taxonomy, and summary of the existing techniques will be valuable for researchers, industry personnel, and standardization bodies.

REFERENCES

[1] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "BinFI: An efficient fault injector for safety-critical machine learning systems," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–23, 2019.

[2] G. Li, K. Pattabiraman, and N. DeBardeleben, "TensorFI: A configurable fault injector for TensorFlow applications," in IEEE International symposium on software reliability engineering workshops (ISSREW), pp. 313–320, IEEE, 2018.

[3] A. Benali Amjoud and M. Amrouch, "Convolutional neural networks backbones for object detection," in Image and Signal Processing: 9th International Conference, ICISP, Marrakesh, Morocco, June 4–6, 2020, Proceedings 9, pp. 282–289, Springer.

[4] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in IEEE symposium on security and privacy (SP), pp. 3–18, IEEE, 2017.

[5] N. Li, W. Qardaji, D. Su, Y. Wu, and W. Yang, "Membership privacy: A unifying framework for privacy definitions," in Proceedings of the ACM SIGSAC conference on Computer & communications security, pp. 889–900, 2013.

[6] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 57–64, IEEE, 2015.

[7] "Globalplatform", ARM system architecture, 2011, [online] available: http://www.globalplatform.org/specificationsdevice.asp."

[8] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, 2016.

[9] W. Li, Y. Xia, and H. Chen, "Research on ARM TrustZone," GetMobile: Mobile Computing and Communications, vol. 22, no. 3, pp. 17–22, 2019.

[10] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "SoK: Security and privacy in machine learning," in IEEE European Symposium on Security and Privacy (EuroS&P), pp. 399–414, IEEE, 2018.

[11] G. S. Kuntla, X. Tian, and Z. Li, "Security and privacy in machine learning: A survey.," Issues in Information Systems, vol. 22, no. 3, 2021.

[12] E. De Cristofaro, "A critical overview of privacy in machine learning," IEEE Security & Privacy, vol. 19, no. 4, pp. 19–27, 2021.

[13] S. Sagar and C. Keke, "Confidential machine learning on untrusted platforms: a survey," Cybersecurity, vol. 4, no. 1, pp. 1–19, 2021.

[14] K. D. Duy, T. Noh, S. Huh, and H. Lee, "Confidential machine learning computation in untrusted environments: A systems security perspective," IEEE Access, vol. 9, pp. 168656–168677, 2021.

[15] "Intel software guard extensions: Intel SGX SDK for Linux OS. http://intel.com. accessed: 2020-06-30."

[16] S. Gueron, "A memory encryption engine suitable for general purpose processors," Cryptology ePrint Archive, 2016.

[17] R. Williams, "AMD confirms it won't opensource epyc's platform security processor code", https://hothardware.com/," 2017.

[18] "https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web."

[19] "https://cloud.google.com/blog/products/identity-security/titan-in-depth-security-in-plaintext."

[20] ""MultiZone secure IoT stack, the first secure IoT stack for RISC-V". Hex Five Security. Hex Five Security, inc. 22 february 2019. retrieved 3 march 2019."

[21] https://hex five.com/, 2019.

[22] S.-C. Wang and S.-C. Wang, "Artificial neural network," Interdisciplinary computing in Java programming, pp. 81–100, 2003.

[23] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.

[24] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in international conference on engineering and technology (ICET), pp. 1–6, IEEE, 2017.

[25] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," Physica D: Nonlinear Phenomena, vol. 404, p. 132306, 2020.

[26] M. P. Ayyar, J. Benois-Pineau, and A. Zemmari, "Review of white box methods for explanations of convolutional neural networks in image classification tasks," Journal of Electronic Imaging, vol. 30, no. 5, pp. 050901–050901, 2021.

[27] K. Mahmood, R. Mahmood, E. Rathbun, and M. van Dijk, "Back in black: A comparative evaluation of recent state-of-the-art black-box attacks," IEEE Access, vol. 10, pp. 998–1019, 2022.

[28] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," Advances in Neural Information Processing Systems, 2014.

[29] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 131–138, IEEE, 2017.

[30] A. Gangal, M. Ye, and S. Wei, "HybridTEE: Secure mobile DNN execution using hybrid trusted execution environment," in Asian Hardware Oriented Security and Trust Symposium (AsianHOST), pp. 1–6, IEEE, 2020.

[31] R. Liu, L. Garcia, Z. Liu, B. Ou, and M. Srivastava, "SecDeep: Secure and performant on-device deep learning inference framework for mobile and IoT devices," in Proceedings of the International Conference on Internet-of-Things Design and Implementation, pp. 67–79, 2021.

[32] T. Elgamal and K. Nahrstedt, "Serdab: An IoT framework for partitioning neural networks computation across multiple enclaves," in 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pp. 519–528, IEEE, 2020.

[33] Z. Liu, Y. Lu, X. Xie, Y. Fang, Z. Jian, and T. Li, "Trusted-DNN: A TrustZone-based adaptive isolation strategy for deep neural networks," in ACM Turing Award Celebration Conference-China, pp. 67–71, 2021.

[34] P. M. VanNostrand, I. Kyriazis, M. Cheng, T. Guo, and R. J. Walls, "Confidential deep learning: Executing proprietary models on untrusted devices," arXiv preprint arXiv:1908.10730, 2019.

[35] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," arXiv preprint arXiv:1806.03287, 2018.

[36] S. P. Bayerl, T. Frassetto, P. Jauernig, K. Riedhammer, A.-R. Sadeghi, T. Schneider, E. Stapf, and C. Weinert, "Offline model guard: Secure and private ML on mobile devices," in Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 460–465, IEEE, 2020.

[37] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang, and J. Song, "Occlumency: Privacy-preserving remote deep-learning inference using SGX," in The 25th Annual International Conference on Mobile Computing and Networking, pp. 1–17, 2019.

[38] M. F. Babar and M. Hasan, "Real-time scheduling of TrustZone-enabled DNN workloads," CPSIoTSec'22, (New York, NY, USA), p. 63–69, Association for Computing Machinery, 2022.

[39] Y. Li, J. Ma, D. Cao, and H. Mei, "Sectum: Accurate latency prediction for ARM-hosted deep learning inference," in IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), pp. 906–916, 2022.

[40] M. S. Islam, M. Zamani, C. H. Kim, L. Khan, and K. W. Hamlen, "Confidential execution of deep learning inference at the untrusted edge with ARM TrustZone," in Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy, CODASPY'23, (New York, NY, USA), p. 153–164, Association for Computing Machinery, 2023.

[41] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "DarkneTZ: Towards model privacy at the edge using trusted execution environments," in Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services, pp. 161–174, 2020.

[42] K. G. Narra, Z. Lin, Y. Wang, K. Balasubramaniam, and M. Annavaram, "Privacy-preserving inference in machine learning services using trusted execution environments," CoRR, vol. abs/1912.03485, 2019.

[43] Y. Xiang, Y. Wang, H. Choi, M. Karimi, and H. Kim, "AegisDNN: Dependable and timely execution of DNN tasks with SGX," in IEEE Real-Time Systems Symposium (RTSS), pp. 68–81, IEEE, 2021.

[44] K. Grover, S. Tople, S. Shinde, R. Bhagwan, and R. Ramjee, "Privado: Practical and secure DNN inference with enclaves," arXiv preprint arXiv:1810.00602, 2018.

[45] A. Kumar, R. Tourani, M. Vij, and S. Srikanteswara, "Sclera: A framework for privacy-preserving MLaaS at the pervasive edge," in IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), pp. 175–180, 2022.

[46] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "TruSpy: Cache side-channel information leakage from the secure world on ARM devices," Cryptology ePrint Archive, 2016.

[47] Y. Jang, J. Lee, S. Lee, and T. Kim, "SGX-bomb: Locking down the processor via Rowhammer attack," in Proceedings of the 2nd Workshop on System Software for Trusted Execution, pp. 1–6, 2017.

[48] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure: SGX cache attacks are practical.," in WOOT, pp. 11–11, 2017.

[49] F. Dall, G. De Micheli, T. Eisenbarth, D. Genkin, N. Heninger, A. Moghimi, and Y. Yarom, "Cachequote: Efficiently recovering long-term secrets of SGX EPID via cache attacks," 2018.

[50] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on Intel SGX," in Proceedings of the 10th European Workshop on Systems Security, pp. 1–6, 2017.

[51] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using SGX to conceal cache attacks," in Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, Proceedings 14, pp. 3–24, Springer.

[52] B. Lapid and A. Wool, "Cache-attacks on the ARM TrustZone implementations of AES-256 and AES-256-GCM via GPU-based analysis," in Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, Revised Selected Papers 25, pp. 235–256, Springer.

[53] B. Lapid and A. Wool, "Navigating the samsung TrustZone and cache-attacks on the keymaster trustlet," in Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I 23, pp. 175–196, Springer.

[54] Y. LeCun, Y. Bengio, et al., "Convolutional networks for images, speech, and time series," The handbook of brain theory and neural networks, vol. 3361, no. 10, 1995.

[55] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size," arXiv preprint arXiv:1602.07360, 2016.

[56] H. Qassim, A. Verma, and D. Feinzimer, "Compressed residual-VGG16 CNN model for big data places image recognition," in IEEE 8th annual computing and communication workshop and conference (CCWC), pp. 169–175, IEEE, 2018.

[57] A. Benali Amjoud and M. Amrouch, "Convolutional neural networks backbones for object detection," in International Conference on Image and Signal Processing, pp. 282–289, Springer.

[58] "Op-TEE, https://www.op-tee.org."

[59] S. Huang, C. Liu, and Z. Su, "Secure storage model based on TrustZone," in IOP Conference Series: Materials Science and Engineering, vol. 490, p. 042035, IOP Publishing, 2019.

[60] J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999.

[61] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont, "Everything you should know about Intel SGX performance on virtualized systems," Proceedings of the ACM on Measurement and Analysis of Computing Systems, vol. 3, no. 1, pp. 1–21, 2019.

[62] R. Freivalds, "Probabilistic machines can use less runnning time, information processing'77 (proc. IFIP congress' 77)," 1977.

[63] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in 25th USENIX Security Symposium, pp. 619–636, 2016.

[64] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution," in 24th USENIX Security Symposium, pp. 431–446, 2015.

[65] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace: Oblivious memory primitives from Intel SGX," Cryptology ePrint Archive, 2017.

[66] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Sanctuary: ARMing TrustZone with user-space enclaves.," in NDSS, 2019.

[67] A. H. Sodhro, S. Pirbhulal, and V. H. C. De Albuquerque, "Artificial intelligence-driven mechanism for edge computing-based industrial applications," IEEE Transactions on Industrial Informatics, vol. 15, no. 7, pp. 4235–4243, 2019.

[68] A. H. Sodhro, Y. Li, and M. A. Shah, "Energy-efficient adaptive transmission power control for wireless body area networks," IET Communications, vol. 10, no. 1, pp. 81–90, 2016.

[69] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural

networks," IEEE journal of solid-state circuits, vol. 52, no. 1, pp. 127–138, 2016.

[70] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in 27th USENIX Security Symposium, pp. 991–1008, 2018.

[71] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "TruSense: Information leakage from TrustZone," in IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, pp. 1097–1105.

[72] A. Waterman and K. Asanovic, "The RISC-V instruction set manual, volume I: Unprivileged ISA document, version 20190608-base-ratified," RISC-V Foundation, Tech. Rep, 2019.

[73] F. Mo, Z. Tarkhani, and H. Haddadi, "SoK: Machine learning with confidential computing," arXiv preprint arXiv:2208.10134, 2022.

[74] Z. Zhang and A. Z. Kouzani, "Implementation of DNNs on IoT devices," Neural Computing and Applications, vol. 32, no. 5, pp. 1327–1356, 2020.

[75] X. Dai, I. Spasić, S. Chapman, and B. Meyer, "The state of the art in implementing machine learning for mobile apps: A survey," SoutheastCon, pp. 1–8, 2020.

[76] S. Mittal, H. Gupta, and S. Srivastava, "A survey on hardware security of DNN models and accelerators," Journal of Systems Architecture, vol. 117, p. 102163, 2021.

MOHAMMAD FAKHRUDDIN BABAR is currently pursuing his Ph.D. at Washington State University (WSU) while also serving as a research assistant at the Cyber-Physical Systems Security Research Lab (CPS2RL). He obtained his B.Sc. in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology in 2017. His research focuses on cyber-physical system security and machine learning.

MONOWAR HASAN (M'2021) is a Computer Science Assistant Professor at Washington State University (WSU). Before joining WSU, he held an Assistant Professor position at Wichita State University from 2021-2022. Dr. Hasan received his Ph.D. in Computer Science from University of Illinois at Urbana-Champaign (UIUC) in 2020 and an M.S. in Electrical and Computer Engineering in 2015 from University of Manitoba (UM). His current research interests include exploring security and resiliency techniques of cyber-physical system domains. Details of Dr. Hasan's research and scholarly activities are available on his academic website (https://monowarhasan.info).