GuardNN: Secure Accelerator Architecture for Privacy-Preserving Deep Learning

Weizhe Hua[†], Muhammad Umar[†], Zhiru Zhang[†], G. Edward Suh[†§*]
[†]Cornell University, Ithaca, NY, USA
[§]Meta AI, Cambridge, MA, USA
{wh399,mu94,zhiruz,gs272}@cornell.edu,edsuh@fb.com

ABSTRACT

This paper proposes GuardNN, a secure DNN accelerator that provides hardware-based protection for user data and model parameters even in an untrusted environment. GuardNN shows that the architecture and protection can be customized for a specific application to provide strong confidentiality and integrity guarantees with negligible overhead. The design of the GuardNN instruction set reduces the TCB to just the accelerator and allows confidentiality protection even when the instructions from a host cannot be trusted. GuardNN minimizes the overhead of memory encryption and integrity verification by customizing the off-chip memory protection for the known memory access patterns of a DNN accelerator. GuardNN is prototyped on an FPGA, demonstrating effective confidentiality protection with ∼3% performance overhead for inference.

1 INTRODUCTION

The past decade has seen unprecedented growth in the use of machine learning (ML). However, the data-intensive nature of ML raises serious concerns for security and privacy. Deep neural networks (DNNs) require collecting, storing, and processing a large amount of personal and private user data. Moreover, DNN computations are often performed in mobile or cloud environments where private data may be exposed or misused. For large-scale deployment of DNNs in privacy-sensitive applications, we need a way to perform DNN computations even in an untrusted environment, with both high performance and strong privacy protection.

A promising approach for providing strong confidentiality and integrity guarantees under untrusted environments is to create a hardware-protected trusted execution environment (TEE), also called an enclave as in Intel SGX [20]. So far TEEs have primarily been studied in the context of general-purpose processors, which cannot provide enough performance and energy efficiency for large-scale ML workloads. This paper proposes to leverage application-specific accelerators to enable high-performance TEEs for ML, and presents a secure DNN accelerator architecture, named *GuardNN*.

To protect sensitive data, GuardNN keeps all confidential information including inputs, outputs, training data, and network parameters (weights) in an encrypted form outside of the trusted hardware boundary, such as an ASIC accelerator chip or an accelerator IP in an SoC. Each accelerator contains a unique private key that can only be used by the accelerator itself. Users can remotely authenticate the accelerator using the corresponding public key and the certificate, and send private inputs and weights encrypted to the accelerator. In this way, GuardNN can ensure that an adversary cannot access private user data and weights even if he/she controls software or has physical access to the accelerator. The secure accelerator can also protect the integrity of ML computation by incorporating remote attestation and off-chip integrity verification.

While GuardNN adopts the high-level approach of today's TEEs, an accelerator TEE needs to address the challenge of providing protection while allowing both a CPU and an accelerator to work together. The accelerator TEE also presents an opportunity to customize protection for a specific application domain to improve both security and performance. For example, while processors can perform arbitrary operations and memory accesses, accelerators only need to support a relatively small set of operations and often have a memory access pattern that is specific to the target application. This application-specific nature of accelerators enables GuardNN to customize its architecture and protection mechanisms for ML, and provide strong security with almost no performance overhead.

The following summarizes the key benefits and insights that GuardNN provides compared to today's general-purpose TEE: 1) GuardNN carefully designs its architecture and instructions to enable confidentiality and integrity protection even when a host CPU that controls scheduling and resource allocation cannot be trusted. This design reduces the trusted computing base (TCB) to just the accelerator. 2) The GuardNN instruction set allows confidentiality-only protection, which is sufficient for privacy-preserving ML, without the complexity and overhead of integrity protection. Regardless of the sequence of instructions, private data are always encrypted outside the accelerator. 3) The accelerator TEE has the potential to provide stronger security compared to CPU TEEs; an accelerator is physically separated from general-purpose cores and has much simpler hardware and software. The memory access pattern and the timing of a DNN accelerator without dynamic pruning is also independent of input and weight values, making GuardNN secure against memory and timing side channels.

We implemented a prototype system based on CHaiDNN [32], an open-source DNN accelerator from AMD Xilinx. The experiments on an AMD Xilinx FPGA demonstrate functional correctness and show that the overhead of memory encryption is negligible. For more detailed analyses, we performed experiments using cycle-level simulations. The simulation results show that GuardNN can guarantee both confidentiality and integrity with small overhead.

[*]Work was done while at Cornell University.

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530439

2 SECURE ACCELERATOR ARCHITECTURE

2.1 Threat Model

We assume that a DNN accelerator can run both inference and training. A scheduler runs on a host CPU and coordinates compute and data movement by communicating with a remote user and issuing commands to the DNN accelerator. The remote user sends inputs and a DNN model, and receives final results.

The goal of a secure DNN accelerator is to protect the confidentiality and the integrity of DNN data and computation in an environment where only the accelerator itself can be trusted. For confidentiality, the secure DNN accelerator aims to protect inputs (inference inputs or training data), prediction results, network parameters, and all intermediate results. On the other hand, we consider the DNN structure as public information and do not hide the structure. For integrity, the secure DNN accelerator aims to detect any unauthorized changes to its state and execution so that a user can verify that the output is the outcome of the given model/input.

The DNN accelerator is trusted and authenticated by the remote user using a unique private key that is only known by the accelerator hardware. The accelerator needs to be designed and fabricated by a trusted manufacturer. The manufacturer also needs to securely embed a private key specific to each accelerator instance, and provide a certificate. We assume that the internal operations and state of the DNN accelerator cannot be directly observed or changed by an adversary whereas anything outside of the accelerator including off-chip memory and a host processor are assumed untrusted.

A typical DNN model has a fixed memory access pattern, and the timing for a given model is agnostic to inputs and weights. In that sense, the GuardNN accelerator is secure against memory and timing side-channel attacks. We do not consider other physical side-channel attacks such as the power and EM side channels.

2.2 Key Insights and Features

Small TCB: DNN accelerators rely on a complex ML software stack for optimizations, scheduling, and resource allocation decisions. Most accelerators cannot efficiently run complex code and need to rely on a CPU for commands. A straightforward design is to protect both CPU and a DNN accelerator in a TCB, where trusted software runs inside a TEE on the host CPU and controls TEE on the accelerator. This design requires not only TEEs on both the CPU and the accelerator, but also a secure communication channel between the two (possibly from multiple vendors), and a remote attestation mechanism that allows users to verify the trustworthiness of a combination of multiple hardware components. In that sense, directly extending today's CPU TEE will lead to a large TCB with complex mechanisms.

Although the ML software stack is complex, the instruction set architecture (ISA) of a DNN accelerator remains simple because the DNN operations can be boiled down to scalar, vector, matrix additions and multiplications and a limited number of non-linear functions. For example, TPU-v1 [12] only has a dozen instructions with five important ones. Based on this observation, we propose to run the ML software on an untrusted host, while restricting the host interface to a limited set that does not leak sensitive information.

GuardNN can ensure confidentiality without trusting a host processor by designing its ISA so that sensitive information is always encrypted no matter which instruction is executed. The outputs are encrypted and can be decrypted only by the remote user who owns the input and the model. Leveraging the typical nature of DNN computations, GuardNN also ensures that the latency of each instruction is independent of secret values. The untrusted host chooses which DNN operations to be performed, but cannot make the accelerator produce outputs in plaintext. The GuardNN design significantly reduces the size of the TCB while allowing the flexibility and performance optimizations provided by the ML software.

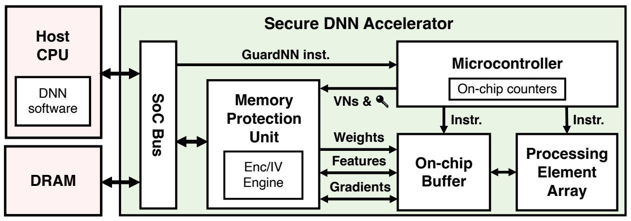


Figure 1: GuardNN architecture overview — The green and red boxes represent trusted and untrusted components, respectively.

Table 1: The security features provided by GuardNN.

Security Function	Mechanism	Threat
Key Generation	True random number generator	Replay/key guessing
Key Exchange	DHE key-exchange protocol	Untrusted host/network
Off-chip Mem. Protection	DNN-optimized memory encryption and integrity verification	Untrusted host/ physical attacks
Restricted Instruction Set	No instruction allows outputting secrets in plaintext	Untrusted host
Remote Attestation	Hashes of input, output, weights, and instructions; Sign the hashes	Untrusted host
Side-channel Protection	The memory access pattern and the timing are independent of secrets	Memory and timing side-channels

Confidentiality-Only Protection: GuardNN can decouple confidentiality and integrity protection, and protect the confidentiality of private data without paying the cost of integrity protection. Regardless of the sequence of instructions, private data are always encrypted outside the accelerator. In addition, the memory access patterns and execution times of DNN accelerators without dynamic pruning [1, 7, 8] are independent of input data values. Hence, the confidentiality guarantees of GuardNN do not depend on the integrity of the instruction sequences and data values. In contrast, the CPU TEEs require integrity protection even for confidentiality; because the trusted software inside the TEE is allowed to output confidential information unencrypted, the integrity of the program must be protected even when only confidentiality is needed.

DNN-Specific Memory Protection: Leveraging the regular and coarse-grained data movement patterns of DNNs, GuardNN removes the need to store version numbers for memory encryption and integrity verification in off-chip memory. This DNN-specific optimizations enable protection with negligible overhead.

2.3 GuardNN Architecture

Here we introduce the GuardNN architecture and its protection mechanisms. Figure 1 shows the high-level block diagram, and Table 1 summarizes the protection mechanisms.

The accelerator needs to be able to establish a secure communication channel with a remote user. For this purpose, a GuardNN accelerator includes a unique private key (SK_{Accel}), a true random

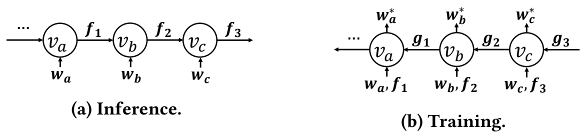


Figure 2: DNN data-flow subgraphs.

number generator, and and a microcontroller. We assume that the user obtains the corresponding public key using a public key infrastructure as in Intel SGX or Trusted Platform Modules (TPMs). GuardNN also introduces an instruction that allows the accelerator to securely exchange a symmetric key ($K_{Session}$) and establish a secure communication channel with the remote user. The user sends DNN model weights, inputs, and outputs through the secure communication channel. GuardNN provides instructions to import encrypted inputs and weights, and produce an encrypted output.

During the execution, GuardNN protects data in external memory using a memory encryption (Enc) engine that encrypts data in DRAM, and an integrity verification (IV) engine that detects unauthorized changes on a read from DRAM. To minimize the performance overhead of memory protection, GuardNN uses a DNN-specific memory protection.

The computation on a DNN accelerator is typically controlled by a software scheduler on a host CPU. While both CPU and accelerators can be protected by a TEE as in a recent GPU TEE design [28], GuardNN enables protection even when the host CPU cannot be trusted. The instruction set is carefully designed to ensure that confidential information is always encrypted outside the accelerator no matter which instruction runs. For side-channel protection, GuardNN requires that the timing and memory access pattern of the accelerator are independent of secret data. This design ensures that confidentiality is protected independent of an instruction sequence. For integrity, GuardNN computes the hashes of inputs and weights when they are imported, and keeps the hash of the sequence of executed instructions and their input arguments, similar to how remote attestation maintains the hash for software state. Then, GuardNN provides an instruction that signs the hashes of each output with the DNN data and instructions using the accelerator's private key so that a user can verify the initial state and the execution.

2.4 Off-chip Memory Protection

2.4.1 Memory Protection Basics. The counter-mode encryption (AES-CTR) is widely used in secure processors [5] to hide AES latency. AES-CTR requires a non-repeating counter value for each encryption, which consists of the physical memory address (PA) of the data block that will be encrypted and a (per-block) version number (VN) that is incremented on each memory write. To prevent data being tampered with by an attacker, integrity verification calculates and stores the MAC of the data value, PA and VN for each data block and checks that MAC on the following read. In addition, to defeat the replay attack, a Merkle tree (i.e., hash tree) [4] is used to verify the MACs hierarchically in a way that the root of the tree is stored on-chip.

2.4.2 DNN-specific Protection. The main overhead of memory protection comes from accessing the off-chip VNs and MACs. Because DNN accelerators are often memory-intensive, the additional metadata accesses can lead to a non-trivial slowdown. Popular ML frameworks often represent the network as a static data-flow graph (DFG) as illustrated in Figure 2 and optimize the graph before execution. Unlike CPUs, DNN accelerators have the same access pattern to a large chunk of memory. A DNN accelerator typically reads/writes the output features of a layer (e.g., f_1, f_2, f_3 in Figure 2a) from/to DRAM the same number of times. This regular memory access pattern allows using only one VN for all output features of a layer. For example, if DNN accelerator only writes the features to DRAM once per layer, the layer number can be used as part of the VN.

In GuardNN, each counter value for memory encryption includes the address of the 128-bit memory block being encrypted/decrypted and a 64-bit VN, and is used as the input to AES-CTR encryption. The VNs are constructed using a few on-chip counters to ensure that the counter values are unique for each encryption. For writing new features, we introduce CTR_{IN} and $CTR_{F,W}$ in the accelerator state to keep track of the number of inputs and the number of times that features are written for the current input. CTR_{IN} is incremented for each new input (**`SetInput`**). $CTR_{F,W}$ is reset on a new input and incremented after each DNN computation instruction (**`Forward`**) that writes output features. The VN for writing features includes both CTR_{IN} and $CTR_{F,W}$.

As the host CPU owns the DFG and controls the scheduling of instructions, the host CPU can easily reconstruct the VN used to write features. For reading the features, GuardNN uses $CTR_{F,R}$ from the CPU to form the VN, and thus avoids tracking the status of the DNN tasks. $CTR_{F,R}$ corresponds to the value of $CTR_{F,W}$ used to write the features. As $CTR_{F,R}$ is only used in decryption, the confidentiality is not broken even if the $CTR_{F,R}$ value is incorrect.

The weights are read-only during inference. Therefore, we can use a constant as the VN for the weights. To allow updating weights during training, GuardNN keeps CTR_W in the accelerator state and keeps track of the number of updates to the weights (**`SetWeight`**). During training, each gradient edge in the DFG has a corresponding feature edge (e.g., f_1 and g_1 in Figure 2b). As the gradients and the features are stored in different memory locations, the gradients can use the VN for the corresponding features.

For integrity protection, MACs still need to be stored in memory. We customize the size of a memory block that each MAC protects to match the data movement granularity of the accelerator. For example, the DNN accelerator that we use for a prototype writes a 512-B chunk to memory at a time.

2.5 GuardNN Instructions

The GuardNN instruction set is designed to be an extension to a DNN accelerator without changing the base instructions. A user can choose if integrity protection is needed when initiating a session.
`GetPK`: Returns the public key (PK) and the certificate (Cert).
`InitSession`: Given a public key from a remote user, the accelerator runs a key exchange protocol to agree on a symmetric session key and establish a secure communication channel with the user. The accelerator also clears all states (keys, data, and hashes), sets a new memory encryption key (K_{MEnc}), resets all counters to zero, and enables memory protection. If integrity protection is enabled, memory integrity verification and hashing of instructions and their operands are also enabled.

`SetWeight` and **`SetInput`**: On **`SetWeight`**, the accelerator imports encrypted weights; these weights are decrypted with the session key ($K_{Session}$) and protected by the accelerator's memory protection in DRAM. Then, the weight counter (CTR_W) is incremented (see Section 2.4). Similarly, on **`SetInput`**, the accelerator imports the encrypted input into its protected memory, and increments the input counter (CTR_{IN}) . For integrity protection, the accelerator also computes the hash of the input/weights for remote attestation.
`ExportOutput`: The accelerator reads an encrypted output from DRAM, and re-encrypts the output with $K_{Session}$ for the user.
`SignOutput`: The accelerator computes a digital signature of the hashes of the input, output, weights, and the sequence of instructions/operands using its private key (SK_{Accel}). By verifying this signature using the corresponding public key, the user can verify that the output was produced by the particular accelerator using the correct initial state and the correct sequence of instructions.
`SetReadCTR`: To reduce overhead and allow complex compiler optimizations, GuardNN relies on the host CPU to determine the VN for reading features. This number is determined based on the network structure and scheduling and does not need to be trusted for confidentiality, as it only affects decryption. Specifically, host CPU sets the $CTR_{F,R}$ value for an address range.

In addition to the instructions listed above, the DNN may require additional preprocessing of the input data. Those preprocessing steps can be handled by the user before sending to the accelerator. Alternatively, GuardNN can also handle most standard image data preprocessing, such as scaling, cropping, clipping, and reflection, by performing the data preprocessing steps as matrix multiplication. Nvidia DALI also proposes to address the problem of the CPU bottleneck by offloading data preprocessing to the GPU.

3 EVALUATION

3.1 Methodology

FPGA Prototype – We implemented a prototype of GuardNN with confidentiality-only protection ($GuardNN_C$) by adding the VN generator, encryption engines (AES-128), and a microcontroller to the CHaiDNN accelerator [32]. We use four different DSP configurations (128, 256, 512 and 1024) with two different precisions (6-bit and 8-bit fixed point) for weights and features. The AES engines are pipelined with a 12-cycle latency. Because AMD Xilinx FPGAs do not currently support secure remote attestation of the bitstream, this prototype is primarily used as a functional demonstration.
Cycle-level Simulation – We use cycle-level simulations to (1) compare the overhead of multiple memory protection schemes, (2) study the overhead for a larger class of DNN models, and (3) evaluate the overhead for DNN training. DNN accelerators are simulated using SCALE-Sim [25], an open-source DNN accelerator simulator from ARM research. The memory accesses are simulated using Ramulator [14] for 16GB DDR4. GuardNN is modeled based on Google TPU-v1 [12], where it contains 64k processing elements (i.e., MAC units) and 24MB on-chip memory.
Benchmarks – We evaluate GuardNN on a variety of DNN architectures — AlexNet, VGG, GoogleNet, ResNet, MobileNet, Vision Transformer (ViT) for image classification, BERT for pretraining language models, DLRM for personalized recommendation, and wav2vec2 for learning speech representation.

Table 2: Throughput and overhead of GuardNN FPGA prototypes — Throughput is reported in frames per second (fps) and overhead (%) is calculated over CHaiDNN baseline.

Throughput (Overhead)	# of DSPs	Network Architecture			
		AlexNet	GoogleNet	ResNet	VGG
$GuardNN_C$ **(8-bit)**	128	51.5 (+0.6)	22.1 (+0.4)	8.1 (+1.2)	2.5 (+0.8)
	256	94.5 (+0.5)	39.4 (+0.5)	14.6 (+1.6)	4.8 (+0.9)
	512	163.6 (+0.3)	64.7 (+1.5)	23.7 (+1.9)	9.0 (+0.6)
	1024	249.4 (**+0.2**)	93.7 (+0.7)	35.3 (+2.4)	15.9 (+0.6)
$GuardNN_C$ **(6-bit)**	128	95.2 (+0.6)	40.4 (+0.5)	14.9 (+1.6)	4.8 (+0.9)
	256	166.3 (+0.5)	67.2 (+0.6)	24.6 (+2.2)	9.1 (+0.9)
	512	258.1 (**+0.3**)	100.2 (+0.8)	37.6 (+2.7)	16.5 (+0.7)
	1024	349.7 (**+0.3**)	128.8 (+1.0)	48.5 (+3.1)	27.6 (+0.6)

3.2 FPGA Prototype Results

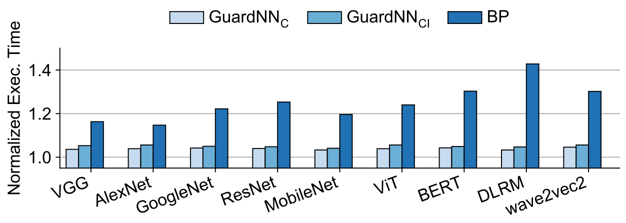
Performance – Table 2 shows the throughput and overhead for various DNN models across several different configurations on an FPGA board. The performance overhead of GuardNN is less than **3.1%** for all eight different configurations on ImageNet. It is worth noting that the overhead comes mainly from the limited throughput of the AES engines. The maximum overhead among the four networks can be further reduced to **1.9%** by increasing the number of AES engines from three (as in Table 2) to four.

We also studied the latency of GuardNN instructions using VGG as an example. GuardNN needs to perform a key exchange and load weights once per session. On the MicroBlaze, the **`GetPK`** and **`InitSession`** (specifically, the ECDHE–ECDSA key-exchange) take 23.1 ms. The key-exchange latency is independent of a network. Importing (decrypting and re-encrypting) weights on **`SetWeight`** takes 19.5ms, 2.2ms, 8.0ms and 43.3ms for AlexNet, GoogleNet, ResNet, and VGG, respectively. For each input, GuardNN adds overhead to import an input and export/sign an output. **`SetInput`** for a single input image only takes 0.1 ms. For the 1000-class output, the **`ExportOutput`** and **`SignOutput`** take 0.01 ms and 4.8 ms, respectively. Thus, the GuardNN instructions incur negligible overhead.
Resource Overhead – In our FPGA prototype with 512 DSPs and 8-bit weights/features, we use an open-source AES-128 IP core that uses 9.0K LUTs and 3.0K FFs. The area overhead of one AES core is 8.2% and 2.6% in LUTs and FFs, respectively. Because the FPGA clock (200 MHz) is much slower than the memory bus clock, three AES engines are needed to match the memory bandwidth used by CHaiDNN. We implemented the microcontroller as a Xilinx MicroBlaze, for which the program can fit within 256KB local memory. The microcontroller's resource usage (overhead) is 2.7K LUTs (2.5%), 2.2K FFs (1.9%), 64 BRAMs (11.0%) & 6 DSPs (0.9%).

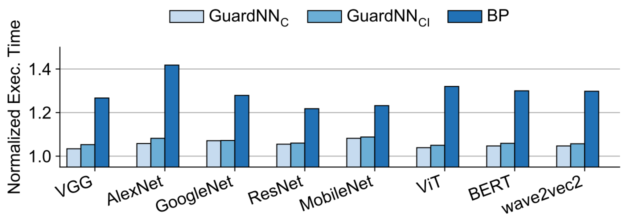
3.3 ASIC Simulation Results

Performance – We study the accelerator performance for four protection schemes: no protection (NP), today's baseline memory protection (BP), and GuardNN with confidentiality-only ($GuardNN_C$) and both confidentiality and integrity protection ($GuardNN_{CI}$). $GuardNN_C$ only performs memory encryption while $GuardNN_{CI}$ perfoms both encryption and integrity verification. For the baseline memory encryption, we implement the recent memory encryption engine (MEE) design from Intel [5] as the state-of-the-art.

As the throughput of a DNN accelerator is often limited by the memory bandwidth, we first compare the memory traffic increase of BP and GuardNN. The memory traffic increase is defined as the ratio



(a) Inference.



(b) Training.

Figure 3: The normalized execution time of the DNN inference and training on different networks models.

between the total number of memory accesses with and without memory protection. BP increases memory accesses by 35.3% on average for inference and by 37.8% for training. The memory traffic increase is larger for training because the training process accesses more data and has more frequent cache evictions in the VN/MAC cache. GuardNN$_{CI}$ increases the memory traffic by **2.4%** and **2.3%** on average for inference and training, respectively.

Figure 3 shows the performance of the baseline protection and the DNN-specific memory protection. BP is 1.25× and 1.29× slower than no protection on average for inference and training. For inference and training, both GuardNN$_C$ and GuardNN$_{CI}$ show much lower performance overhead than BP. The average overhead of GuardNN$_{CI}$ is **1.05%** for inference and **1.07%** for training. GuardNN$_C$ further reduces the overhead to **1.04%** for inference and **1.05%** for training. The results demonstrate that GuardNN can support secure DNN processing with negligible overhead over the baseline accelerator with no protection.

ASIC Power/Area Overhead – The power and area overhead of GuardNN is expected to be low for an ASIC design. The additional area mainly comes from the AES engines, which are used for encryption and integrity verification. A previous study [26] shows that a low-power AES engine only consumes 0.0031 mm^2 in area and 3.85 mW in power, while achieving a 991 Mbps throughput at 875 MHz in 28nm ASIC. In contrast, the area and power consumption of TPU-v1 [12] (also in 28nm) are 331 mm^2 and 75 W, respectively. Notably, TPU-v1 runs at 700 MHz and has a peak memory bandwidth of 272 Gbps. In order to match the memory bandwidth of TPU-v1, we can instantiate 344 AES engines, which only result in 0.3% area and 1.8% power overhead. We can also use a smaller number of high-performance AES engines, which will likely have similar overall overhead.

3.4 Comparison with Alternatives

Table 3 compares GuardNN with other approaches for privacy-preserving deep learning. For a CPU TEE, the table shows the performance for a simulated CPU TEE with unlimited protected memory to represent the ideal case of CPU TEEs. Today's Intel SGX limits its protected memory to 128 MBs, which leads to 5-20× performance overhead for DNNs [13].

GuardNN$_{CI}$ provides confidentiality and integrity guarantees for VGG with only 5% overhead, while the simulated CPU TEE adds more than 60% overhead on the same benchmark. By leveraging hardware-based protection and high performance accelerators, GuardNN achieves three orders of magnitude higher performance and energy efficiency compared to alternatives. As discussed in Section 2, GuardNN also has a small TCB thanks to the simplicity of DNN accelerator design and protection mechanisms. The lines of code (LoC) for GuardNN prototype is 21.8k in total — 9k LoC for the baseline accelerator, 8.3k LoC for the customized protection, and 4.5k LoC for new instructions (firmware on a microcontroller). MPC-based approaches also have a relatively small size of TCB and can be implementation with tens of thousands of LoC. However, their performance overhead is significantly higher than GuardNN. While MPC-based approaches offer secure and easy-to-deploy options for less performance-demanding use cases, secure accelerators appear to be the most promising solution for large-scale, high-throughput use cases.

4 RELATED WORK

Privacy-Preserving Deep Learning – GuardNN provides hardware-based protection for DNN inference and training in an untrusted environment. Alternatively, homomorphic encryption (HE) and secure multi-party computation (MPC) can provide stronger protection on today's hardware by performing all computations in an encrypted format. While HE and MPC provide strong cryptographic guarantees without trusting remote hardware or software, they come with significant overhead compared to the baseline with no protection [3, 15, 19, 21, 24, 29]. A recent work [23] proposes to reduce the latency of HE-based DNN inference to hundreds of milliseconds using specialized hardware. GuardNN provides a design point that provides hardware-based security with much higher performance compared to the HE/MPC-based solutions.

TEEs provide hardware-protected execution environments where confidentiality and integrity are ensured even under an untrusted OS or physical attacks. Recent studies showed that DNN computations can be protected using Intel SGX [13, 18, 27] but with non-trivial overhead of memory protection in SGX. Today's processor-based TEEs are also limited by the performance of a general-purpose processor. Recent studies [11, 28] proposes to extend today's TEE by including a GPU. The GPU-based TEEs enable much higher DNN performance compared to general-purpose processors, but require both a CPU and a GPU to be protected inside a TEE. Telekine [10] further improves the security of GPU TEEs by translating the application's GPU API calls into data-oblivious commands.

Recent work [30, 33, 34] proposes to build FPGA/ASIC TEEs as accelerators are often far more energy-efficient than GPUs and widely used for high-throughput tasks such as inference. TNPU [17] concurrently proposes a tree-less off-chip memory protection for DNN accelerators, similar to our DNN-specific memory protection. GuardNN allows a smaller TCB and lower overhead by carefully desining its instruction set for an untrusted host and an option for confidentiality-only protection.

Table 3: Comparison between different privacy-preserving ML approaches — The throughput is measured in giga operations per second (GOPs) and the energy efficiency is reported in giga operations per second per Watt (GOPs/W). For GuardNN accelerators, we show the number of PEs, the size of on-chip SRAM, and the clock frequency. The power of the GuardNN is estimated based on TPU-v1.

Metrics		Methods				
		CPU TEE (Simulated)	DELPHI MPC [21]	CrypTFLOW2 MPC [22]	GuardNN$_{CI}$ (Simulated)	GuardNN$_{C}$ (FPGA)
Hardware		CPU 1 core@3.0 GHz	Intel Xeon 4 cores@3.7 GHz	Intel Xeon 4 cores@3.7 GHz	64k PEs/24 MBs @0.7 GHz	512 PEs/3 MBs @0.2 GHz
Workloads	Network	VGG-16	ResNet-32	ResNet-32	VGG-16	VGG-16
	Dataset	ImageNet	CIFAR-100	CIFAR-100	ImageNet	ImageNet
Perf.	Throughput	0.81	0.02	0.18	3221.57	139.23
	Overhead (×)	1.61	~1000	~100	1.05	1.01
Energy	Power (W)	~60	130	130	~40	~15
	Efficiency	0.01	0.002	0.0001	80.5	9.3
TCB	Components	CPU	MPC protocol	MPC protocol	Accelerator	Accelerator
	Lines of code	Millions [16]	35.1k	53.7k	10-100s of thousands	21.8k

GuardNN proposes a new approach to enable secure DNN computation using accelerators and shows that secure accelerators have a potential to provide higher security with a negligible performance and area overhead compared to the general-purpose platforms by customizing its architecture and protection for DNNs.

Side-channel Attacks and Protection – A variety of side-channel attacks have been shown to work against DNN accelerators. Memory and timing side-channels have been used to infer the network structure of an accelerator with encrypted weights [6, 9]. GuardNN has a fixed memory access pattern and execution time, and is secure against memory and timing side-channels. Physical side-channel attacks such as power and electromagnetic side-channel attacks have been used to retrieve the input image [31] or recover the network topology and weights [2]. If strong protection against power and EM side-channel attacks is necessary, GuardNN needs to be extended with additional countermeasures.

5 CONCLUSION

This paper proposes a secure DNN accelerator, named GuardNN. Application-specific accelerators provide strong isolation from a CPU with complex software stack and also enable protection to be customized for DNNs to improve both security and performance. An FPGA prototype shows that the GuardNN can protect common DNN models with minimal (~3%) performance overhead.

6 ACKNOWLEDGMENT

We thank the anonymous reviewers for their constructive feedback. At Cornell, Weizhe Hua and Muhammad Umar are supported in part by NSF Award CCF-2007832, ECCS-1932501, and CCF-2118709. Weizhe Hua is also supported in part by the Facebook fellowship.

REFERENCES

[1] J. Albericio et al. 2016. Cnvlutin: Ineffectual-neuron-free Deep Neural Network Computing. In *ISCA*.

[2] L. Batina et al. 2019. CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. In *USENIX Security*.

[3] N. Dowlin et al. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *ICML*.

[4] B. Gassend et al. 2003. Caches and hash trees for efficient memory integrity verification. In *HPCA*.

[5] S. Gueron. 2016. Memory Encryption for General-Purpose Processors. In *S&P*.

[6] W. Hua et al. 2018. Reverse Engineering Convolutional Neural Networks Through Side-channel Information Leaks. In *DAC*.

[7] W. Hua et al. 2019. Boosting the Performance of CNN Accelerators with Dynamic Fine-Grained Channel Gating. In *MICRO*.

[8] W. Hua et al. 2019. Channel Gating Neural Networks. In *NeurIPS*.

[9] W. Hua et al. 2022. Reverse Engineering CNN Models using Side-Channel Attacks. In *IEEE Design & Test*.

[10] T. Hunt et al. 2020. Telekine: Secure Computing with Cloud GPUs. In *NSDI*.

[11] I. Jang et al. 2019. Heterogeneous Isolated Execution for Commodity GPUs. In *ASPLOS*.

[12] N. P Jouppi et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA*.

[13] K. Kim et al. 2020. Vessels: Efficient and Scalable Deep Learning Prediction on Trusted Processors. In *SoCC*.

[14] Y. Kim et al. 2016. Ramulator: A Fast and Extensible DRAM Simulator. In *CAL*.

[15] N. Kumar et al. 2020. CrypTFlow: Secure TensorFlow Inference. In *S&P*.

[16] D. Lee et al. 2020. Keystone: An Open Framework for Architecting Trusted Execution Environments. In *EuroSys*.

[17] S. Lee et al. 2022. TNPU: Supporting Trusted Execution with Tree-less Integrity Protection for Neural Processing Unit. In *HPCA*.

[18] T. Lee et al. 2019. Occlumency: Privacy-Preserving Remote Deep-Learning Inference Using SGX. In *MobiCom*.

[19] G. Lloret-Talavera et al. 2022. Enabling Homomorphically Encrypted Inference for Large DNN Models. In *IEEE TC*.

[20] F. McKeen et al. 2016. Intel Software Guard Extensions (Intel SGX) Support for Dynamic Memory Management Inside an Enclave. In *HASP*.

[21] P. Mishra et al. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *USENIX Security*.

[22] D. Rathee et al. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS*.

[23] B. Reagen et al. 2021. Cheetah: Optimizing and Accelerating Homomorphic Encryption for Private Inference. In *HPCA*.

[24] T. Ryffel et al. 2022. AriaNN: Low-Interaction Privacy-Preserving Deep Learning via Function Secret Sharing. In *PET*.

[25] A. Samajdar et al. 2020. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim. In *ISPASS*.

[26] W. Shan et al. 2019. A 923 Gbps/W, 113-Cycle, 2-Sbox Energy-efficient AES Accelerator in 28nm CMOS. In *VLSI*.

[27] F. Tramer et al. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *ICLR*.

[28] S. Volos et al. 2018. Graviton: Trusted Execution Environments on GPUs. In *OSDI*.

[29] S. Wagh et al. 2021. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. In *PET*.

[30] X. Wang et al. 2019. NPUFort: A Secure Architecture of DNN Accelerator Against Model Inversion Attack. In *CF*.

[31] L. Wei et al. 2018. I Know What You See: Power Side-Channel Attack on Convolutional Neural Network Accelerators. In *ACSAC*.

[32] Xilinx. 2018. CHaiDNN-v2: HLS based Deep Neural Network Accelerator Library for Xilinx Ultrascale+ MPSoCs. https://github.com/Xilinx/CHaiDNN.

[33] M. Zhao et al. 2022. ShEF: Shielded Enclaves for Cloud FPGAs. In *ASPLOS*.

[34] J. Zhu et al. 2020. Enabling Rack-scale Confidential Computing using Heterogeneous Trusted Execution Environment. In *S&P*.