Triton: Software-Defined Threat Model for Secure Multi-Tenant ML Inference Accelerators

Sarbartha Banerjee
sarbartha@utexas.edu
University of Texas at Austin
Austin, Texas, USA

Shijia Wei
shijiawei@utexas.edu
University of Texas at Austin
Austin, Texas, USA

Prakash Ramrakhyani
prakash.ramrakhyani@arm.com
ARM Research
Austin, Texas, USA

Mohit Tiwari
tiwari@austin.utexas.edu
University of Texas at Austin
Austin, Texas, USA

ABSTRACT

Secure machine-learning inference is essential with the advent of multi-tenancy in machine learning-as-a-service (MLaaS) accelerators. Model owners demand the confidentiality of both model weights and architecture, while end users want to protect their personal data. Moreover, ML models used in mission-critical applications like autonomous vehicles or disease classification need integrity protection. While hardware trusted execution environments (TEE) [4, 41] provide data confidentiality and integrity, they face two challenges in the adoption for ML inference. First, TEEs are susceptible to numerous side channels, arising from resource sharing in multi-tenant systems. Second, the performance overhead of these TEEs is often proportional to the secret data size, making them unattractive for data-intensive real-time inference.

The diverse deployment threats further complicate these challenges. For instance, compared to time-sharing execution, multi-tenant accelerators must assume a larger attack surface with adversaries monitoring or tampering with on-accelerator resources. Some inference process sensitive inputs while others compute on public inputs. As a result, existing TEE designs often adopt a single, perhaps the most restrictive threat model, which overburdens many secure ML inference deployments.

To address the challenges in adopting TEEs for secure ML inference, we introduce the *Triton* TEE framework. *Triton* tailors threat models to each deployment with low overhead while mitigating side-channel leakages. *Triton* achieves this by offering an interface to define fine-grained secrets in an ML model or input, along with the attacker observation capabilities. *Triton* framework generates code for a custom threat model for each application based on its security requirements. The security policy of each secret is embedded in the instruction to convey the security guarantee to the hardware. The expressive threat model and secret declaration can reduce the secure ML inference overhead from 64% to 6% across different multi-tenant deployments.

HASP '23, October 29, 2023, Toronto, Canada

ACM ISBN 979-8-4007-1623-2/23/10.
https://doi.org/10.1145/3623652.3623672

CCS CONCEPTS

• **Security and privacy** → **Hardware security implementation**; Tamper-proof and tamper-resistant designs; • **Computer systems organization** → Architectures; **Neural networks**.

KEYWORDS

Secure hardware, ML accelerator, TEE, Threat model

ACM Reference Format:
Sarbartha Banerjee, Shijia Wei, Prakash Ramrakhyani, and Mohit Tiwari. 2023. *Triton*: Software-Defined Threat Model for Secure Multi-Tenant ML Inference Accelerators. In *Hardware and Architectural Support for Security and Privacy 2023 (HASP '23), October 29, 2023, Toronto, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3623652.3623672

1 INTRODUCTION

ML inference has entered a new phase with the emergence of chatGPT [6] and other attention-based language models [56]. In addition to speech recognition [25], image classification [27, 39] and recommendation systems [22], ML inference is being used in content generation [19], sophisticated chatbots [15], named entity recognition [44] and even labeling data for training new models [53]. The endless opportunities of ML inference have encouraged many organizations to train their own private models. Platforms like Amazon sagemaker [1] have created an interface to host these trained models to provide inference service. Multiple tenants share large ML accelerators, to cater to the quality-of-service (QoS) of multiple customers as well as, ensure higher resource utilization to amortize cost. Prior work show how *temporal* [14] and *spatial* [20] sharing of ML inference accelerators improve execution QoS through efficient resource utilization. *Temporal* sharing interleaves model layers from multiple tenants to improve overall QoS. *Spatial* sharing, on the other hand, concurrently executes multiple tenants by partitioning accelerator resource. Several side-channels [10, 31] emerge from multi-tenant ML inference, as attackers can closely monitor victim execution through compute port contention, memory bandwidth variation, and stale private data in the scratchpad. Co-executing attackers can alter internal control queues or victim scratchpad data to compromise ML inference integrity.

Researchers have proposed secure ML inference to ensure the confidentiality and integrity of ML accelerator inference. Cryptographic techniques [21, 28, 32, 47], used for secure ML inference, incur more than 10× slowdown and is impractical for large models.

ML accelerator TEEs [29, 41, 58] can provide these security guarantees. However, the execution overhead becomes proportional to the model size, which has reached 100 billion parameters [12]. Moreover, they are prone to side-channels, that mainly stem from direct and indirect on-chip shared resources between applications. Multi-tenant ML inference, needs on-chip micro-architectural isolation, in addition to memory protection. While traditional micro-architectures have dynamic data paths and buffers invisible to the software, providing on-chip isolation is a monumental task. Prior work [17] partitions CPU micro-architecture including LLC caches and MSHRs to provide secret isolation. *Triton* framework uses similar partitioning during spatial accelerator sharing. However, a temporal tenant can use all hardware partitions as it does not have co-executing tenants. Moreover, all the micro-architectural components in an ML accelerator can be explicitly controlled by the compiler, giving absolute control on each secret buffer or scratchpad storage. The key challenge, however, in providing such a defense, is the large overhead introduced by these TEEs, due to the large secret data volume of ML inference workloads.

The first key observation, is that, not all data in a trained ML model is a secret. Due to the large data corpus needed to train models, many of the ML models trained today use transfer learning [55]. A public trained model, designed to perform the same task is taken from repositories like Huggingface [3]. This model is fine-tuned [23] by either retraining some of the final layers, or changing the model topology by replacing some layers. This training method is cost-effective and requires a smaller dataset. These models have only a few proprietary layers that needs to be protected. Same is true for inference input, where some of the image pixels or specific word tokens in a sentence query is secret information.

The second observation is that, the attacker observations vary with accelerator deployments. For instance, on-chip resource isolation is essential for *spatial* sharing but not for *temporal* sharing. This is because multiple tenants co-execute in a spatially shared accelerator. Moreover, several ML accelerators use on-chip high-bandwidth memory (HBM). These memories may not be susceptible to physical attacks and might not require data integrity protection. An attacker might not have access to memory traffic to decipher the model layer size.

Given these observations, we propose *Triton*, a flexible secure ML accelerator framework, that enables the developer to define fine-grained secrets and the security policies for each application. A developer can define, each layer of a trained model as either *private* or *public* data. The same is true for inference input, where *private* data can be defined at a pixel granularity for images or token granularity for texts. The user can additionally define security properties (eg. confidentiality, integrity etc.) explicitly for each secret data. Taint-analysis in compiler propagates *private* inputs to mark intermediate variables with corresponding security properties. Micro-architectural isolation is guaranteed between between *private* and *public* data, belonging to the same tenant, as well as, across tenants. The developer can further define deployment characteristics like single or multi-tenant accelerator sharing. Such fine-grained accelerator and data declarations tailor security guarantees to only the *private* data reducing the performance penalty for secure ML inference. The key contributions of *Triton* framework are the following:

- We introduce a secure ML inference accelerator framework that allows deployment time threat model and secret definition and generates code based on the protection guarantees requested by a developer.
- The secure accelerator hardware, not only protect against off-chip memory confidentiality and integrity attacks, but also other side-channels including model layer dimensions in the memory bus and victim execution utilization.
- Security policy is communicated via novel ISA extensions to the hardware , allowing distinct security guarantees for each data structures.
- *Triton* evaluates multiple threat models with different performance and security policies. The performance overhead ranges from 9% to 64% during temporal and 6% to 62% during spatial sharing compared to non-secure execution.
- We perform three real-world case studies, where only a portion of inference input or ML model is a secret and demonstrate how distinct declaration of security policies provide additional performance benefit.

2 BACKGROUND

We provide a background on model secrets, data secrets, ML accelerator architecture, and TEEs.

2.1 ML model secrets

The method with which, the model owner trains a model, determines model confidentiality. The two secrets associated with a model are the model weight values and the model topology. All the layer weights are confidential for a model, when it is trained from scratch. In addition, the sequence and layer types along with their size can be compromised by an attacker [10, 31].

Another common method to train a model is through transfer learning [55]. The last few layers of a public model, trained on a similar task, are replaced with confidential layers. The model weights of these confidential layers are trained on a smaller dataset, targeting a specific task. The secrets, in this case, are the weights and topology of only the confidential layers. Since the initial layer shape and weights are frozen during training, they retain their public value. Another flavor of transfer learning, does not change the model topology, but only retrain the weights of the last few layers. In this case, the entire model topology is public, while the last few weights are confidential. The model, however, can be entirely integrity protected to prevent tampering with inference.

2.2 Inference data secrets

Inference data can be entirely or partially confidential to the data owner. An X-ray image, used for disease classification can be entirely confidential to a patient, while, only the face pixels in a group photo can be private information. Similarly, a few word tokens (like the credit card number) can be a secret in chatbot conversations. The query size can be representative of the confidential query in a language translation model.

Current ML TEEs do not have a framework to protect partial secrets, which leads to the entire input to be defined as a secret. Declaration of partial secrets along with the guarantees, lowers the inference latency for many applications.

2.3 ML inference accelerator architecture

ML accelerators are built on decoupled-access execute (DAE) architecture [54] with specialized systolic array [46] dataflow. Several popular accelerators like Google TPU [7] and others [24, 40, 48] have an array of multiply-accumulate (MAC) units to perform matrix-matrix or vector-matrix computation. Double buffered scratchpads perform memory loads in parallel to computation. Data dependencies between the load/store and compute units are managed with dependency queues. These accelerators have a dedicated ISA, with `load` and `store` instructions for memory operations and `gemm` and `alu` for computation as in VTA [48]. To improve the resource utilization of large compute units, several works [9, 14, 20] have extended these accelerators to support multi-tenant inference. Model inferences can be time-shared in a single accelerator hardware (*Temporal Sharing*), or run multiple inferences concurrently by splitting the accelerator (*Spatial Sharing*).

2.4 Security guarantees of TEEs

Hardware TEEs create an isolated execution environment for processing confidential data. A device attestation by a root of trust ensures a benign platform. Authenticated encryption schemes like AES-GCM [18] provide confidentiality and integrity to secret data in memory. TEEs are designed for both CPUs [2, 4] and accelerators [29, 42, 58, 60] and provide architectural isolation to secrets.

3 MOTIVATION

3.1 Flexible threat model for ML TEEs

The security guarantees provided by a TEE are fixed at hardware design time. For instance, Intel SGX has hardware support to provide data confidentiality and integrity. MI6 [11] uses cache partitioning and micro-architectural isolation with *purge* instructions. Similarly, multi-tenant accelerators need hardware support for memory protection, accelerator partitioning, resource invalidation and memory bandwidth obfuscation for eliminating side-channels [10, 31]. As discussed in §2.1, the security policy of a *private* model is application-specific. TEEs do not distinguish between *public* and *private* data and provide a fixed security guarantee to all secrets.

Moreover, the partial or entire model confidentiality also depend on the training strategy. ML model trained with transfer learning can have a small percentage of *private* data. A TEE with flexible security policy and secret declaration reduces the performance overhead of secure ML inference. However, a framework has to ensure that there is no dataflow from *private* to *public* domain. We introduce taint-tracking security passes to ensure there is no dataflow across the security domains. Once, all the *private* data is isolated, specialized ISA extensions convey the security policy for each data structure. The hardware enables only the required defence mechanisms producing a secure yet low latency inference.

3.2 Deployment specific TEE requirements

The attack surface change across ML inference deployments. For instance, many of the cloud accelerators have HBM [35] memory, which might not require integrity protection. The memory bandwidth utilization by a *private* model is not accesible to attackers in cloud service provider deployments. Moreover, on-chip resource isolation is only necessary when the tenants co-execute in a spatially shared accelerator. *Triton* framework generates code, taking into consideration, these deployment characteristics. Such observations are critical for low latency real-time ML inference scenarios.

4 THREAT MODEL

Triton has a flexible threat model, chosen by the developer at runtime. However, to understand the defense capabilities of *Triton*, this section discusses the most restrictive threat model: *Triton* is running a secret model to process secret inputs in a spatially multi-tenant accelerator. The attacker has the strongest observation capability in this threat model. We assume secret data is stored encrypted and authenticated in the main memory. And attackers cannot read or tamper with secret data in memory. Moreover, data replay attacks are prevented with the help of integrity counters.

An attacker cannot obtain secret data value or size information from snooping the memory bus. Encrypted data traffic flowing through the read and write bus is shaped by a constant traffic shaper, hiding bandwidth variations. The attacker's observation is limited to a fixed-size data transaction on each data bus throughout the tenant execution time. The attacker cannot contend for memory bandwidth due to fixed bandwidth allocation for each tenant.

A spatially co-located attacker cannot read or tamper with victim secrets in the scratchpad or compute buffers. The compute units and the associated data and control buffers are partitioned among tenants to prevent port contention or other timing side channels.

At execution termination, all secret data is invalidated by writing zeros to the scratchpad, and the data buffers before a new tenant is provisioned. Attackers cannot mount use-after-free attacks to read stale secret data or use timing channels during context switches.

5 *TRITON* DESIGN OVERVIEW

5.1 Secure ML accelerator primitives

Triton is designed on TVM [13] compiler. The generated code is evaluated on a scale-sim [51] simulator.

5.1.1 Software primitives. An ML inference application is written with TVM compiler APIs. The TVM parser is augmented to include application primitives discussed in §5.2. Existing compiler passes are used for dataflow graph generation, optimization and code generation. Additional security passes like taint-tracking, memory flushing, integrity metadata generation and configuration register writes are added to the compiler and discussed in §5.3. TVM runtime is used to generate the model binary to be deployed in hardware. The addition of security policies to each instruction is described in §5.3.2.

5.1.2 Hardware primitives. The generated binary is simulated using a cycle-accurate scalesim simulator, where is it loaded into an instruction queue. An instruction decoder routes memory instructions to the memory controller, compute requests to the issue queue and invalidate instructions to clear micro-architectural resources. Ramulator [37] is integrated to simulate the memory transactions. The transactions are sent to the ramulator and the data is ready for computation after the data arrival time.

5.2 Application APIs:

Triton introduces new pragmas for threat model definition and arguments for secret policy for data structures. Listing 1 shows

the addition of two matrices of size 1024×1024. The first matrix (mod) is defined entirely as a secret while a part of the second vector (inp) is secret. Lines 1-2 import the numpy and the TVM tensor expression (te) library. The platform configuration is defined in lines 5-6. Security configurations are specified through *sec_pragma*(). In this case, the platform is *spatially* shared with multiple tenants with data integrity verification performed at 1024B granularity. Memory bandwidth limit, scratchpad allocation and number of compute units reserved is further specified by *res_pragma*().

```
1  import numpy as np
2  from tvm import te
3  n = 1024
4  #Define the deployment configuration
5  sec_pragma(exec_mode ="spatial", inte_gran=1024)
6  res_pragma(mem_bw=0.25, spad=0.25, comp=0.25)
7  #Secret model weights and topology
8  mod = te.placeholder((n,n), name="mod",
9      encr=1, inte=1, shap=1)
10 #Define secret input region
11 x_range = np.array(start=50, stop=60)
12 y_range = np.array(start=20, stop=30)
13 #Partial input secret
14 inp = te.placeholder((n,n), name="inp",encr=1, inte=1,
       shap=1, x_conf=x_range, y_conf=y_range)
15 #Performing vector addition
16 out = te.compute(mod.shape,
17 lambda i: mod[i] + inp[i], name="out")
```

Listing 1: Sample code for addition of two secret matrices

Line 8 defines a secret vector (mod), which requires data encryption, integrity verification and memory traffic shaping. We augment the `te.placeholder` TVM api to include these security properties through new arguments (`encr,inte,shap`). This enables the developer to define each variable with customized security policies. The same API is used to define partial input secrets as shown in line 14. Here, the sub-matrix whose range is defined by *x_range* and *y_range*, is *private*. The remaining matrix is a *public* variable. The compiler guarantees the required security guarantees to the *private* variable region. Finally, the matrix addition is carried out by the `te.compute` function. Such explicit security declaration for each variable enables *Triton* to provide a runtime threat model customized for each application.

5.3 Compiler Enhancements

TVM compiler is enhanced to **(1)** Generate hardware configuration register writes for pragma declaration; **(2)** Compiler passes to provide *private* data with the specified security guarantees.

5.3.1 Accelerator configuration: The *Triton* compiler transforms the *sec_pragma*() and *res_pragma*() declarations into hardware configuration register writes. These registers are programmed to partition the micro-architecture and reserve requested resources for each tenants. Some of the values like `inte_gran` further customizes the granularity of integrity protection for every application. Traffic shaper can also be configured to define a traffic shape for each application. Currently, we only consider a constant bandwidth traffic shaper and leave other shapes for future research.

5.3.2 Triton ISA:. New instructions communicate data security requirements to hardware. *Triton* ISA is based on VTA [48] instructions, that work with the TVM runtime. Basic instructions include `load` and `store` for memory operations, and `gemm` and `alu` instructions for compute operations. Three flag bits are added to memory instructions corresponding to data declaration arguments (`encr,inte,shap`). The hardware decoder enables the encryption, integrity and the traffic shaper unit based on these flags. The compute instructions have a constant-time execution flag to avert side-channels from data-driven optimizations on secret data. We leave the evaluation of data-driven optimizations to future work. Finally, a new `invalidate` instruction flushes buffers and scratchpads holding secret data.

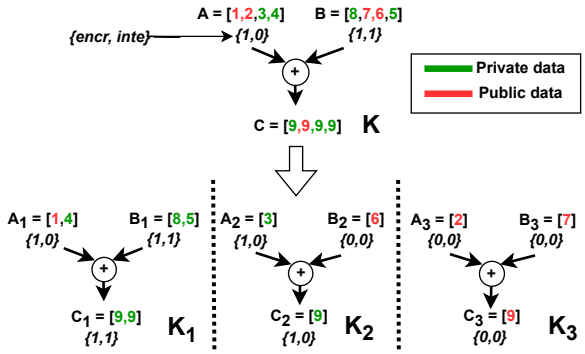

Figure 1: Secret propagation through taint-tracking of a vector addition kernel (K). The kernels are split into sub-kernels (K_1, K_2, K_3) based on the security properties of the output sub-vector (C_1, C_2, C_3). The $\{encr, inte\}$ values are for private (green) data sub-vector. It is $\{0,0\}$ for the public (red) data.

5.3.3 Secret taint-tracking: TVM compiler transforms an ML model into a dataflow graph. *Triton* adds a taint-tracking mechanism to propagate security properties of computation outputs. Fig. 1 shows a vector addition between two variables A and B. The green indices are *private* with $\{encr, inte\}$ defined by the developer. The red indices denote unencrypted *public* data. *Triton* compiler splits the computation based on the security property of the input variables. C_1 inherits the union of security properties of A_1 and B_1. In this case, C_1 is confidential and needs integrity protection. Similarly, C_2 is computed on encrypted A_2 and public B_2. Therefore, C_2 only requires only data confidentiality. C_3 is propagated as a *public* variable. Taint-tracking makes sure, each *private* data element is appropriately safeguarded in micro-architecture and scheduled separately from its *public* counterpart.

5.3.4 Backward taint-propagation: After a forward pass of secret taint-tracking, a single operation is broken into multiple sub operations. While fine-grained taint propagation provides the exact security property of an intermediate variable (C), this might not efficiently run in a high-throughput ML accelerator. A backward taint propagation pass reduces compute fragmentation in certain cases. Let us assume that C_2 in Fig. 1 is consumed in a subsequent operation with C_1. In this case, the operation output will have the properties of C_1. In this scenario, the backward pass would merge K_1 and K_2 kernels to generate an output, that requires both encryption and integrity. This way, the number of kernel splits are reduced, in favor of better kernel resource utilization.

5.3.5 Compute scheduling: Compute scheduling includes breaking large matrices into smaller tiles to fit in the accelerator. Each layer is profiled for different tile sizes, given the resource declaration and the lowest latency ones are chosen. Certain layers are split into a number of parallel data graphs by the taint propagation step. Private kernels requiring integrity protection need additional metadata transactions. So, the public kernels are executed first, during which the counters and MAC are prefetched for private kernels. This kernel re-ordering reduces pressure of loading private data. Moreover, the scheduler strives to minimize the secret data movement and harness maximum data locality inside the accelerator. This reduces the overhead associated with decryption, integrity check and secret invalidation during context switch.

5.3.6 Memory traffic scheduling: Memory requests are shaped to protect input or model size if a variable has $shap = 1$. As we will discuss in §5.4.3, traffic shaper hardware sends fake transactions at times of low demand. An attacker should not be able to distinguish between real and fake transactions based on data size or memory response delay. The compiler splits each data load or store into equal sized fixed transactions and places them into adjacent banks to ensure no leaks from bank access pattern.

5.3.7 Code generation: The *Triton* compiler generates the application binary with instructions in §5.3.2. The flag bits in each `load` and `store` instructions are populated to reflect the variable security property. Moreover, for data operation needing integrity verification ($inte = 1$), additional `load` instructions perform counter and MAC loads from memory. The taint-tracking provides a list of secret variables, which are invalidated from scratchpad and compute buffers with `invalidate` instructions. Finally, the resource allocation and the threat model is sent as a part of binary and is written to accelerator configuration registers during bootstrapping.

5.4 Hardware Implementation:

In this section, we discuss the hardware defense mechanisms required in an accelerator to enforce the security properties.

5.4.1 Accelerator interface: A secure co-processor initiates a TLS session with the remote tenant and sends a device attestation report. Upon device authenticity verification, the tenant sends the ML model binary through a secure channel. The model instructions are loaded in main memory, while the resource and security configurations are written to registers. A unique tenant ID is generated to map resource partitions. Encryption and integrity keys, received from the binary is stored in key storage, tagged to the tenant ID.

This secure co-processor, serving as the accelerator interface, has a trusted firmware, which is verified by secure boot. We assume the firmware to have a limited API, including the capability of running a TLS session and generate an attestation report to the user. The secure co-processor, should have a TPM module [38] to sign the attestation report. Our simulator leverages shef [58], which implements a similar accelerator interface for bootstrapping tenants. Side-channels associated with the secure co-processor and verification of the trusted firmware is out-of-scope and is a topic for future work.

5.4.2 Encryption and Integrity unit: Triton hardware uses AES-GCM [18] to perform authenticated encryption on user-defined *private* data. The *encr* and *inte* flags of each decoded `load` and `store` instruction decides the operation of this unit. The integrity unit has separate buffers for integrity counters and MAC. The size of these buffers are listed in Table 1. The integrity metadata (counters and MAC) is loaded to respective buffers with explicit `load` instructions. For spatially shared tenants, these buffers are partitioned among tenants through configuration registers. The encryption and the integrity unit can be individually bypassed for instructions when the *encr* and *inte* flags are zero respectively. The AES encryption/decryption and MAC computation are simulated with timing delays mentioned in [8]. The integrity unit waits for the load transactions of MAC and counter values to be loaded before adding the MAC computation delay. The data is ready in scratchpad, only after the integrity verification is complete.

5.4.3 Memory traffic shaper: To protect the model topology or secret input size, a constant memory traffic shaper is deployed at the accelerator memory interface for each read and write bus. Demand requests with from each tenant is queued in a request buffer. The requests are sent out to the memory at regular intervals, based on the requested memory bandwidth. The traffic shaper sends memory transactions to memory banks in a round-robin manner. If a transaction corresponding to the next bank is unavailable, it sends a fake transaction to that bank. The memory controller follows a closed row strategy which enforces a uniform transaction response time. The traffic shaper is only activated if a transaction in the request queue has $shap = 1$. If there is a bank conflict across tenants, only one real transaction is sent and the other tenants send a fake transaction. The fake transactions do not access the memory banks and the response is ignored by the traffic shaper.

5.4.4 Scratchpad invalidation: The accelerator scratchpad is split into $4kB$ regions. A table tracks the occupation status of each scratchpad regions. Each entry has several fields: a tenant ID, a valid bit, a read-write bit and a secret bit. When a tenant is bootstrapped, it reserves scratchpad regions and the ID entry is populated. Assignment of tenant ID ensures access-control of scratchpad regions during multi-tenant execution. For temporal sharing, all the table entries are owned by the active tenant. During context switch, the scratchpad regions having the secret bit set is invalidated to prevent side-channels. Such scratchpad entries are written with zeros after secret data invalidation.

5.4.5 Partitioning logic: Triton hardware contains several buffers for holding instruction, integrity metadata, control dependencies, pending memory requests etc. All these buffers are statically partitioned to support multiple tenants. A tenant, requiring more resource can reserve more than one such partitions. The number of executing tenants, at any instant, is kept secret from the attacker. The accelerator interface never over-subscribes an accelerator, preventing micro-architectural contentions.

6 EVALUATION

6.1 Platform

Triton compiler is implemented using TVM [13] while the accelerator hardware is simulated using a cycle-accurate scale-sim [51] integrated with ramulator [37] as the memory interface. The spatial and temporal configurations for ML inference accelerator are listed

Parameter	Temporal Mode	Spatial Mode
Tenants	1	4
Memory type	High-Bandwidth Memory (HBM)	
Memory Bandwidth	128 GB/s	32 GB/s
Compute units	256×256	64×64
Input Scratchpad	6144 KB	1536 KB
Weight Scratchpad	6144 KB	1536 KB
Output Scratchpad	2048 KB	512 KB
Counter Buffer	128 entries	32 entries
MAC Buffer	64 entries	16 entries

Table 1: *Triton* simulator parameters for temporal and spatial modes. The spatial mode is simulated with four concurrent tenants sharing equal sized partitions.

in Table 1. The models are context-switched at layer boundary during *temporal* sharing, while four instances of the same benchmark is run during *spatial* sharing. The accelerator configuration for the temporal mode simulates a Google TPU [34] pod. The ramulator is configured only as a HBM for simplicity.

The system is evaluated with a divese set of benchmarks including image classification models – **Alexnet** [39] and **Resnet-50** [27]; image segmentation models used for object detection – **FasterRCNN** [50], a recommendation and personalization system – **DLRM** [49], a gaming bot – **AlphaGoZero** [52], the encoding layer of a LSTM based language translation – **Translate** [5], a basic transformer with text embeddings, the encoder and decoder for language translation – **Transformer** [56].

6.2 Threat model based latency

Different security properties like data confidentiality, integrity and data structure can be independently defined in *Triton* framework with the `incr`, `inte` and `shap` arguments. Fig. 2 shows the ML inference latency for nine threat models for each application run in temporal and spatial sharing modes. The first three bars (green) run a private ML model with private inputs; The next three bars (cyan) run a private ML model with public inputs; The last three bars (magenta) run a public ML model with private inputs. The first of the three bars in each category has all *incr*, *inte* and *shap* as 1 for private data. The second bar represents a case, when the private data does not require data integrity ($incr = 1$, $inte = 0$, $shap = 1$). The third bar only protects data confidentiality ($incr = 1$), while *inte* and *shap* are 0. Precise security definition can drastically improve the latency overhead associated with ML inference. Without the *Triton* framework, all of these nine cases would have the overhead associated with the first green bar, which is, on average, 62% during temporal sharing and 64% for spatial sharing over non-secure execution. However, if the user is only concerned with private input confidentiality, the geometric mean of performance overhead is only 9% for temporal and 6% for spatial sharing (last bar). The performance overhead of other scenarios also decrease, compared to the first bar and represent threat models, that are relevant in different real-world deployments.

The three bars of same color provides insight on overhead associated with different security guarantees. The performance difference between the first and the second bar is the integrity overhead, while the difference between the second and the third bar arise from the traffic shaper. $Transformer$, *DLRM* have a large number of small kernels, each getting delayed by integrity checks. The overhead decreases considerably when they are executed in a platform, not needing integrity guarantees. *Alexnet* is compute-intensive and is least affected by integrity check and the traffic shaper. Traffic shaper overhead is high for application with strided and non-uniform data access as is evident in *Resnet*50, $FasterRCNN$ and $Translate$. Additional fake transaction is inserted to request data from contiguous banks delaying real transactions.

The latency overhead depends on the security policy programmed by the developer. The performance overhead for the most stringent threat model ($incr = 1$, $inte = 1$, $shap = 1$) is still high. The temporal overhead of 62% comes from micro-architectural cleanup at each context switch. The evaluation assumes tenant context switch at each layer boundary. In reality, tenants get context-switched at a much lower rate, especially for small layers. Some of the large workloads do not fit well in the smaller spatial partitions. The developer may choose to allocate multiple spatial partitions to reduce the performance overhead of such applications. Choosing the appropriate resource allocation with $res_pragma()$ is necessary to meet the latency QoS and can be configured during compilation.

6.3 Case Studies

The following three case studies showcase the usefulness of partial secret declaration for ML models and inference input.

[C1] ML inference with a transformer model, whose last few layers are confidential.

[C2] A FasterRCNN model which performs multiple tasks, among which, a subset of task is performed by a private model.

[C3] A Resnet50 image classification of a partially secret input.

6.3.1 [C1] Partially secret transformer. Transformer networks are large and require a large data corpus to train from scratch. Hence, a public transformer network, trained on the same task is imported by the model owner from Huggingface [3] or other repositories. The model owner modifies the last few layers and re-trains them on a smaller target dataset. The weights of initial layers are kept same as the base model and is *public* data. This method is widely used to train new transformer models, where the last few trained layers constitute the *private* layers. A transformer network consists of an encoder and a decoder block, followed by a linear and softmax layer to generate output probabilities. The decoder of the Transformer [56] consists of two multi-head attention layers followed by a feed forward network. In this case study, the decoder feed forward network, the linear and the softmax layers are fine-tuned by the model owner. The encoder and the multi-head attention layers of the decoder is *public* and its weights are used as-is from the base model. We infer a public sentence with this private model.

Without the *Triton* framework, the model has to be defined entirely as secret to protect model weights and structure. The latency overhead is 83% for temporal and 38% for spatial sharing as shown in the first bar of Fig. 3. The remaining bars define the model with only the last feed-forward, the linear and the softmax layers as secret with different security policies. The temporal overhead ranges from $7 - 21\%$ while the spatial overhead ranges from $8 - 15\%$. This large reduction is due to the public declaration of initial layers, which includes the large embedding layer.

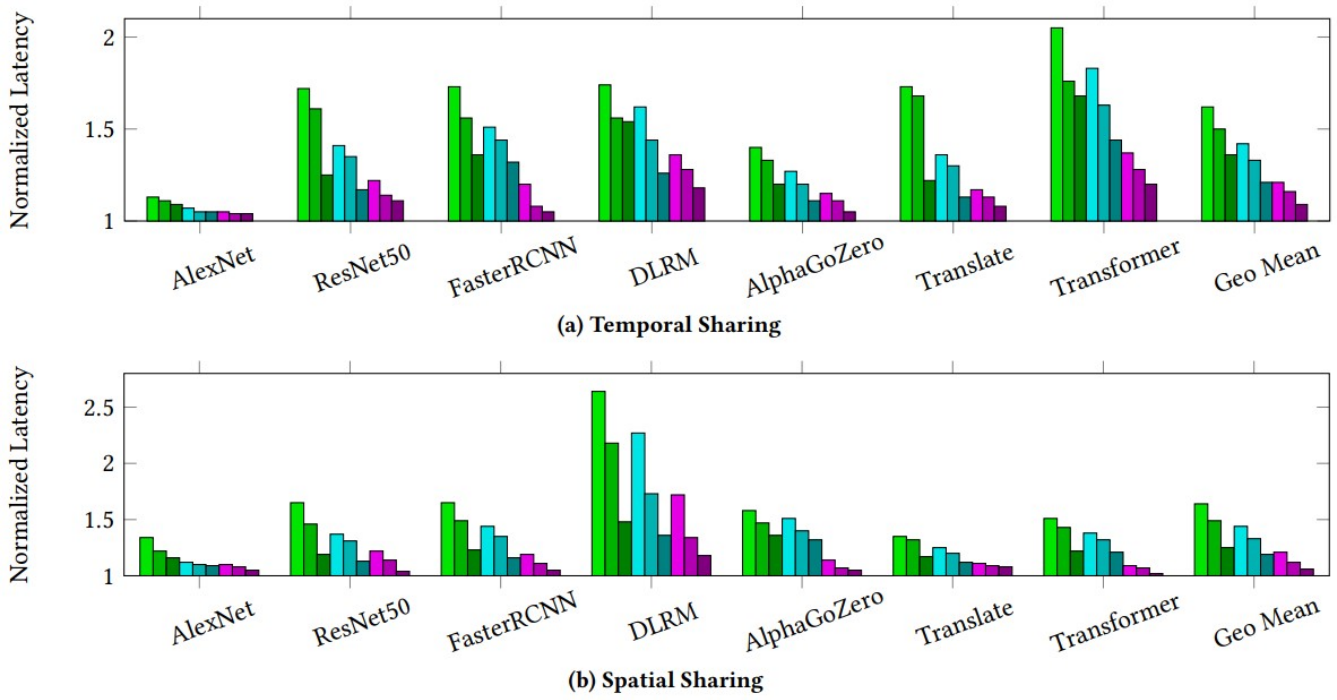

Figure 2: Latency of different threat models normalized to non-secure execution. The first three (green) bars show private input inference with a private model. The next three (cyan) bars public input inference with a private model. The last three (magenta) bars show private input inference with a public model. The private data of the first bars in each group has $\{encr = 1, inte = 1, shap = 1\}$. The second and third bars have $\{encr = 1, inte = 0, shap = 1\}$ and $\{encr = 1, inte = 0, shap = 0\}$.

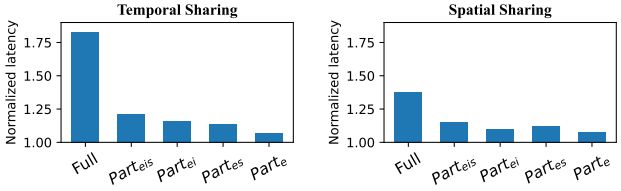

Figure 3: The *Full* bar shows the inference latency without the partial secret declaration of a Transformer model trained with transfer learning. The *Part* bars show overhead with partial model secret declaration. The suffixes *e,i,s* represent encr, inte and shap values are set.

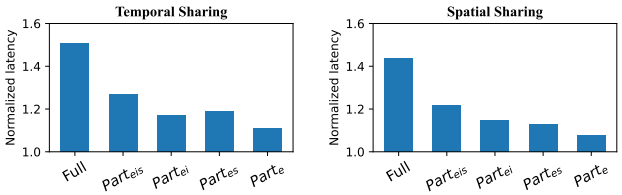

Figure 4: Latency reduction of *FasterRCNN* with a *public* initial feature extractor and a *private* classifier. *Triton* framework enables partial model *private* declaration with customized security policies (*e,i,s*) for multi-stage models.

6.3.2 [C2] Multi-stage model inference. Models like *FasterRCNN* performs object classification in multiple stages. The first stage extracts feature regions in an image, while a next stage classifies them. The model owner uses a *public* feature extractor and a *private* classifier for a *FasterRCNN* model in this case study. Fig. 4 shows the overhead reduction, when the model owner specifies only the secret classifier. The partial secret model declaration is key in making secure ML inference real-time in object detection and image segmentation models. Many of these models use a single initial stage for finding interesting regions, which are fed to multiple classifiers.

6.3.3 [C3] Partial secrets in ML inference inputs. In this case study, the user performs an image classification with a public *Resnet*50 on a passport photo. The pixels containing the face data is defined *private*, while the background and the torso is *public*. In this experiment, 40% of the pixels, containing the face information is marked secret. Since, the size of a passport photo is *public*, using the traffic shaper is irrelevant. The user is only interested in the confidentiality and integrity of the image. Hence, the *incr* and the *inte* bits of the face pixels are set for the input image. The temporal overhead for *Resnet*50 is 18%. However, with the input *private* region definition capability, the overhead reduces to 8%. The spatial overhead reduces from 20% to 12% This reduction is in part due to less decryption and integrity verification cycles and in part, due to less average occupancy of metadata buffers. The majority of overhead comes from the later layers were the *private* data is scattered across a large number of feature vectors.

7 SECURITY ANALYSIS

The security of *Triton* framework depends on **(1)** Trusted tenant bootstrapping; **(2)** The isolation provided by hardware defenses; **(3)** The propagation of security policies from application to hardware.

Trusted tenant bootstrapping is ensured through secure booting the co-processor, device authenticity through remote attestation, transfer of data over a secure channel and accelerator configuration and data load by a trusted firmware as mentioned in §5.4.1.

7.1 Execution isolation guarantees

The confidentiality of *private* data is preserved through encryption in main memory and on the memory bus. However, it is stored as plaintext in scratchpad and compute buffers. These on-chip microarchitectural resources are inaccessible in temporal mode by other tenants and partitioned in spatial mode. Before context switch, the *private* data is invalidated and zeros are written in place. This protects both secret value and occupancy in scratchpad and buffers.

Data integrity is ensured by storing additional counters and MAC for data with $inte = 1$. The integrity is verified before starting computation, hence cannot be tampered in main memory or the memory bus. Access control in scratchpad prevents tenants to access/modify data beyond its allocation. The resource partitioning is done by the trusted firmware, which ensures no overlap among tenants during resource allocation.

Side-channels arising from resource contention is prevented through on-chip buffer, compute resource and scratchpad partitioning. The entire datapath from memory to accelerator and back is partitioned among tenants in spatial sharing. This includes memory request queue, integrity metadata buffers, scratchpads, compute buffers, and even the compute units. The control units including the instruction queue, the buffers holding decoded instructions and the dependency queues are also partitioned. This eliminates any contention between the attacker and the victim tenant. The closed-row policy in memory prevents timing channels due to bank conflicts. Memory traffic utilization information is obfuscated by the traffic shaper in both read and write buses. The shaper requests data from consecutive banks hiding the access. The traffic shaper sends a fake transaction on behalf of one of the tenants, when there is a bank conflict between tenants. Resource utilization is prevented in temporal sharing mode by invalidating the entire accelerator during context switch.

7.2 Compiler isolation guarantees

The compiler passes need to ensure that the *private* data is isolated from *public* data. *Triton* ISA ensures propagation of security policy of each variable to the hardware. The secret taint tracking pass labels any data either directly defined by the user or generated from an operation with at least one *private* input as *private*. This condition is sufficient in *private* data declaration. Moreover, the security policy of any output variable is the union of the input policies. Execution isolation between private and public data is performed through compute and memory scheduling. Decoupling the private data memory and compute scheduling from the public data isolates attacker visibility on the private section of execution. We assume *Triton* compiler is a part of our trusted computing base (TCB) and will not perform any malicious operations.

8 RELATED WORKS

While protections of DNN models have been explored on Intel SGX [36, 45], GPUs [33, 57, 60], or combination of both [26], recent work pioneered building accelerator-based TEEs [29, 30, 42, 43, 58]. Most are focused on memory protection for DNN workloads. TNPU, GuardNN, and MGX [29, 30, 42, 43] proposed tree-free integrity verification exploiting DNN-specific data access patterns. However, *Triton* identifies several use cases, where the secret space can be reduced with precise definition of security properties. This enables *Triton* to provide additional side-channel protections like partitioning, traffic shaping and secret invalidation, while still providing low performance overhead. ShEF [58] explored a framework on secure boot, remote attestation, and isolated execution for FPGA-TEEs, while we focus on ML inference accelerator ASIC providing configurable defence mechanisms based on deployment time threat model and secret policy declaration of model layers and inference inputs.

Several works [9, 14, 20] explored the performance impact and the overall QoS improvement of multi-tenant accelerators. *Triton*, on the other hand, discuss the security implications of ML accelerator multi-tenancy. Micro-architectural isolation is studied for CPUs, where prior work uses hardware partitioning [11, 17] and traffic shaping [16, 59]. *Triton* explores partitioning and shaping techniques for ML accelerators and provide runtime configuration to these isolation mechanisms.

9 CONCLUSION

In this paper, we present *Triton*, a hardware-software design that supports configurable threat models for multi-tenant ML inference accelerators. *Triton* flexibly provides security policies like data confidentiality, integrity, traffic shaping on demand basis to each data structure. By tailoring hardware security primitives to user-desired threat models, *Triton* brings down the inference latency within 10% for both spatial and temporal sharing modes. The ability to declare a subset of model layers or inference input as *private* provides additional performance benefits. Deployment time threat model declaration makes *Triton* framework adaptable to the diverge secure ML inference deployments.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback. This work is funded by SRC ACE JUMP 2.0 program and Intel RARE program.

REFERENCES

[1] [n. d.]. Amazon sagemaker. https://aws.amazon.com/sagemaker/.
[2] [n. d.]. ARM Trustzone. https://www.arm.com/technologies/trustzone-for-cortex-a.
[3] [n. d.]. HuggingFace. https://huggingface.co.
[4] [n. d.]. Intel Software Guard Extensions. https://software.intel.com/en-us/sgx.
[5] [n. d.]. LSTM in keras. https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html.
[6] [n. d.]. openAI chatgpt. https://chat.openai.com.
[7] [n. d.]. System Architecture of TPUv4. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm.
[8] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Hector Montaner, Prakash Ramrakhyani, Francesco Regazzoni, and Andreas Sandberg. 2020. Protecting Memory Contents on ARM Cores. In *Ninth Real World Crypto Symposium (RWC '20)*. https://rwc.iacr.org/2020/slides/Avanzi.pdf

[9] Eunjin Baek, Dongup Kwon, and Jangwoo Kim. 2020. A Multi-Neural Network Acceleration Architecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. https://doi.org/10.1109/ISCA45697.2020.00081

[10] Sarbartha Banerjee, Shijia Wei, Prakash Ramrakhyani, and Mohit Tiwari. 2021. Bandwidth Utilization Side-Channel on ML Inference Accelerators (2021). arXiv:2110.07157

[11] Thomas Bourgeat, Ilia Lebedev, Andrew Wright, Sizhuo Zhang, Arvind, and Srinivas Devadas. 2019. Mi6: Secure enclaves in a speculative out-of-order processor. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*.

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners (2020).

[13] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA. https://www.usenix.org/conference/osdi18/presentation/chen

[14] Yujeong Choi and Minsoo Rhu. 2020. Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[15] Robert Dale. 2016. The return of the chatbots (2016).

[16] Peter W Deutsch, Yuheng Yang, Thomas Bourgeat, Jules Drean, Joel S Emer, and Mengjia Yan. 2022. DAGguise: mitigating memory timing side channels. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[17] Jules Drean, Miguel Gomez-Garcia, Thomas Bourgeat, and Srinivas Devadas. 2023. Citadel: Side-Channel-Resistant Enclaves with Secure Shared Memory on a Speculative Out-of-Order Processor (2023).

[18] Morris J Dworkin. 2007. *Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac*. National Institute of Standards & Technology.

[19] William Fedus, Ian Goodfellow, and Andrew M Dai. 2018. Maskgan: better text generation via filling in the_ (2018).

[20] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, et al. 2020. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[21] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*.

[22] Carlos A Gomez-Uribe and Neil Hunt. 2015. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)* 6, 4 (2015).

[23] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. 2019. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*.

[24] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016).

[25] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. 2014. Deep speech: Scaling up end-to-end speech recognition (2014).

[26] Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. 2021. DarKnight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision*. Springer.

[28] Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li. 2021. Model Protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Transactions on Dependable and Secure Computing* 19, 6 (2021).

[29] Weizhe Hua, Muhammad Umar, Zhiru Zhang, and G. Edward Suh. 2022. GuardNN: Secure Accelerator Architecture for Privacy-Preserving Deep Learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. New York, NY, USA. https://doi.org/10.1145/3489517.3530439

[30] Weizhe Hua, Muhammad Umar, Zhiru Zhang, and G. Edward Suh. 2022. MGX: Near-Zero Overhead Memory Protection for Data-Intensive Accelerators. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. New York, NY, USA. https://doi.org/10.1145/3470496.3527418

[31] Weizhe Hua, Zhiru Zhang, and G. Edward Suh. 2018. Reverse Engineering Convolutional Neural Networks Through Side-channel Information Leaks. In *Proceedings of the 55th Annual Design Automation Conference* (San Francisco, California) *(DAC '18)*. ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3195970.3196105

[32] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and fast secure {two-party} deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*.

[33] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J Rossbach, and Emmett Witchel. 2020. Telekine: Secure Computing with Cloud GPUs. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*.

[34] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE.

[35] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. Hbm (high bandwidth memory) dram technology and architecture. In *2017 IEEE International Memory Workshop (IMW)*. IEEE.

[36] Kyungtae Kim, Chung Hwan Kim, Junghwan" John" Rhee, Xiao Yu, Haifeng Chen, Dave Tian, and Byoungyoung Lee. 2020. Vessels: Efficient and scalable deep learning prediction on trusted processors. In *Proceedings of the 11th ACM Symposium on Cloud Computing*.

[37] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Comput. Archit. Lett.* 15, 1 (jan 2016). https://doi.org/10.1109/LCA.2015.2414456

[38] Steven L Kinney. 2006. *Trusted platform module basics: using TPM in embedded systems*. Elsevier.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (may 2017). https://doi.org/10.1145/3065386

[40] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. 2020. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. *IEEE Micro* 40, 3 (2020).

[41] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Dawn Song, and Krste Asanović. 2019. Keystone: A framework for architecting tees (2019).

[42] Sunho Lee, Jungwoo Kim, Seonjin Na, Jongse Park, and Jaehyuk Huh. 2022. TNPU: Supporting Trusted Execution with Tree-less Integrity Protection for Neural Processing Unit. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. https://doi.org/10.1109/HPCA53966.2022.00025

[43] Sunho Lee, Seonjin Na, Jungwoo Kim, Jongse Park, and Jaehyuk Huh. 2022. Tunable Memory Protection for Secure Neural Processing Units. In *The 40th International Conference on Computer Design (ICCD) 2022*.

[44] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. 2020. A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering* 34, 1 (2020).

[45] Yuepeng Li, Deze Zeng, Lin Gu, Quan Chen, Song Guo, Albert Zomaya, and Minyi Guo. 2021. Lasagna: Accelerating Secure Deep Learning Inference in SGX-Enabled Edge Cloud. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. New York, NY, USA. https://doi.org/10.1145/3472883.3486988

[46] Richard J Lipton and Daniel Lopresti. 1985. A systolic array for rapid string comparison. In *Proceedings of the Chapel Hill Conference on VLSI*.

[47] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*. IEEE.

[48] Thierry Moreau, Tianqi Chen, Ziheng Jiang, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. VTA: An Open Hardware-Software Stack for Deep Learning (2018).

[49] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems (2019).

[50] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks (2015).

[51] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. https://doi.org/10.1109/ISPASS48437.2020.00016

[52] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016).

[53] Oriane Siméoni, Gilles Puy, Huy V Vo, Simon Roburin, Spyros Gidaris, Andrei Bursuc, Patrick Pérez, Renaud Marlet, and Jean Ponce. 2021. Localizing objects with self-supervised transformers and no labels (2021).

[54] James E Smith. 1982. Decoupled access/execute computer architectures. In *ACM SIGARCH Computer Architecture News*, Vol. 10. IEEE Computer Society Press.

[55] Lisa Torrey and Jude Shavlik. 2010. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need (2017).

[57] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: trusted execution environments on GPUs. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*.

[58] Mark Zhao, Mingyu Gao, and Christos Kozyrakis. 2022. Shef: Shielded enclaves for cloud fpgas. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[59] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. 2017. Camouflage: Memory traffic shaping to mitigate timing attacks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.

[60] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Boyan Zhao, Zhongpu Wang, Yuhui Zhang, Jiameng Ying, Lixin Zhang, and Dan Meng. 2020. Enabling Rack-scale Confidential Computing using Heterogeneous Trusted Execution Environment. In *2020 IEEE Symposium on Security and Privacy (SP)*. https://doi.org/10.1109/SP40000.2020.00054