SecureLoop: Design Space Exploration of Secure DNN Accelerators

Kyungmi Lee
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
kyungmi@mit.edu

Mengjia Yan
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
mengjiay@mit.edu

Joel S. Emer
Massachusetts Institute of Technology / NVIDIA
Cambridge, Massachusetts, USA
jsemer@mit.edu

Anantha P. Chandrakasan
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
anantha@mit.edu

ABSTRACT

Deep neural networks (DNNs) are gaining popularity in a wide range of domains, ranging from speech and video recognition to healthcare. With this increased adoption comes the pressing need for securing DNN execution environments on CPUs, GPUs, and ASICs. While there are active research efforts in supporting a trusted execution environment (TEE) on CPUs, the exploration in supporting TEEs on accelerators is limited, with only a few solutions available [18, 19, 27]. A key limitation along this line of work is that these secure DNN accelerators narrowly consider a few specific architectures. The design choices and the associated cost for securing these architectures do not transfer to other diverse architectures.

This paper strives to address this limitation by developing a design space exploration tool for supporting TEEs on diverse DNN accelerators. We target secure DNN accelerators equipped with cryptographic engines where the cryptographic operations are closely coupled with the data movement in the accelerators. These operations significantly complicate the scheduling for DNN accelerators, as the scheduling needs to account for the extra on-chip computation and off-chip memory accesses introduced by these cryptographic operations, and even needs to account for potential interactions across DNN layers.

We tackle these challenges in our tool, called SecureLoop, by introducing a *scheduling search engine* with the following attributes: 1) considers the cryptographic overhead associated with every off-chip data access, 2) uses an efficient modular arithmetic technique to compute the optimal authentication block assignment for each individual layer, and 3) uses a simulated annealing algorithm to perform cross-layer optimizations. Compared to the conventional schedulers, our tool finds the schedule for secure DNN designs with up to 33.2% speedup and 50.2% improvement of energy-delay-product.

MICRO '23, October 28-November 1, 2023, Toronto, ON, Canada

ACM ISBN 979-8-4007-0329-4/23/10.
https://doi.org/10.1145/3613424.3614273

CCS CONCEPTS

• **Computer systems organization** → **Neural networks**; **Data flow architectures**; • **Security and privacy** → **Security in hardware**.

KEYWORDS

Trusted execution environment, neural networks, accelerator scheduling

ACM Reference Format:
Kyungmi Lee, Mengjia Yan, Joel S. Emer, and Anantha P. Chandrakasan. 2023. SecureLoop: Design Space Exploration of Secure DNN Accelerators. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28-November 1, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3613424.3614273

1 INTRODUCTION

Deep neural networks (DNNs) are increasingly deployed in security-critical applications that process sensitive user information or make high-stakes decisions. However, the security threats exploiting hardware-level vulnerabilities can undermine the privacy and integrity necessary for such applications. For example, prior work has shown that the confidentiality of the DNN models and training data can be leaked via bus snooping attacks and cold boot attacks [11]. Moreover, the integrity of the DNN models can be tampered with via data corruption attacks and RowHammer attacks [16, 17, 31, 44, 52], leading to malfunctioning DNNs that generate either extremely low accuracy or biased output on the assigned tasks.

An appealing solution to these security threats is to provide a trusted execution environment (TEE) for DNN computation. There have been extensive research efforts in supporting TEEs on CPUs, including the commercialized solutions by major chip vendors and various open-source solutions from academia, such as Intel SGX [7] and Keystone [26]. These solutions often rely on cryptographic operations (in software or hardware) to perform encryption and authentication for off-chip data accesses [9, 10, 37, 39, 45, 51]. However, as those solutions target general-purpose applications, they cannot match the high data-intensity nature of DNN applications, let alone being used in DNN accelerators. As such, researchers have been working on customizing TEE solutions for DNN accelerators [18, 19, 27]. These schemes leverage the structured and predetermined data access patterns of DNN accelerators and derive a coordination plan between data movement and cryptographic

operations. As a result, the cryptographic operations and the data movement in DNN accelerators become closely coupled.

However, there exists one key limitation of all existing works. Recent years have witnessed innovations in DNN accelerators with various designs and deployment setups, ranging from high-performance data centers to low-power edge devices [5, 6, 13, 22, 30]. DNN accelerator architectures can vary significantly in terms of dataflow, PE, and on-chip buffer organizations. Unfortunately, the existing works on secure DNN accelerator designs only considered a few specific architectures [22] as their baseline designs, and it is difficult to generalize the cost of securing those designs to other diverse DNN accelerators with distinct performance goals and area/energy budgets. This paper aims to address this limitation and develop a design space exploration tool for secure DNN accelerators. We identify several challenges in developing such a tool, especially related to identifying the optimal scheduling of the workload.

Challenges Secure DNN accelerators need to include on-chip cryptographic engines that perform encryption and authentication operations. To ensure data integrity, a cryptographic hash is introduced that is associated with each block of data (called an authentication block) and is used to verify the integrity of the data before performing any computation on it. For data confidentiality, this process requires the decryption of data flowing from DRAM to on-chip buffers and the encryption of data flowing in the opposite direction. When fetching a unit of data from DRAM to on-chip buffers, we need to fetch the whole authentication block containing this unit of data with its corresponding hash. Upon writing the data back to DRAM, a new hash needs to be computed based on the whole block of data and written back together with the data. The cryptographic operations described above introduce extra on-chip computation and additional off-chip memory accesses.

When designing a design space exploration tool for secure DNN accelerators, in addition to counting the performance overhead of cryptographic operations, we need to tackle an important yet unexplored research challenge. The challenge arises when *the authentication block is not fully aligned with the tiling of data* (tiles are the unit for data movement between memory levels in DNN accelerators). Such misalignment of the authentication block and the tiles leads to fetching redundant data for the purpose of performing cryptographic authentication rather than DNN computation.

This challenge is further exacerbated by cross-layer dependency among the layers in a DNN. Specifically, the output feature map of one layer is used as the input feature map to the next layer, and hashes will be computed and associated with fixed authentication blocks when the output feature map is generated. Since tile assignment is done independently for consecutive layers in traditional DNN scheduling, misalignment in tiling between one layer's output feature map and the next layer's input feature map introduces additional challenges for assigning authentication blocks. Furthermore, cross-layer dependency due to authentication blocks also implies that the schedules of consecutive layers are intertwined, exponentially increasing the search space for scheduling.

This Paper In this work, we present a framework for design space exploration of secure DNN accelerators, for systematic investigation of the performance, area, and energy trade-off for supporting a TEE in different DNN accelerator designs. A fair comparison among different designs requires a scheduling algorithm that can elicit the best possible performance of an accelerator design for a given DNN workload [4, 8, 15, 20, 32].

At the core of our framework is a *scheduling search engine* with three steps. First, we start by augmenting a baseline DNN scheduler with the capability to take the performance and energy overhead of the cryptographic engines into account. Next, we formulate the authentication block assignment problem into a mathematical problem that can be analytically solved with computationally-efficient algorithm and figure out the optimal authentication block size for each datatype and layer. Finally, we solve cross-layer optimization of the overall scheduling using simulated annealing, a heuristic-based search algorithm, to trade-off between search time and the quality of results.

We implement SecureLoop framework upon an existing scheduling tool, Timeloop [32]. We show that, compared to the baseline scheduler that targets traditional DNN accelerators without cryptographic support, our cryptographic-engine-aware scheduler can find better schedules for secure DNN designs with up to 33.2% speedup and 50.2% improvement of energy-delay-product. We use our tool to perform a thorough design space exploration across multiple DNN workloads, and we derive the area versus performance trade-offs for different secure accelerator designs, providing insights on which designs can be the Pareto front of this trade-off curve.

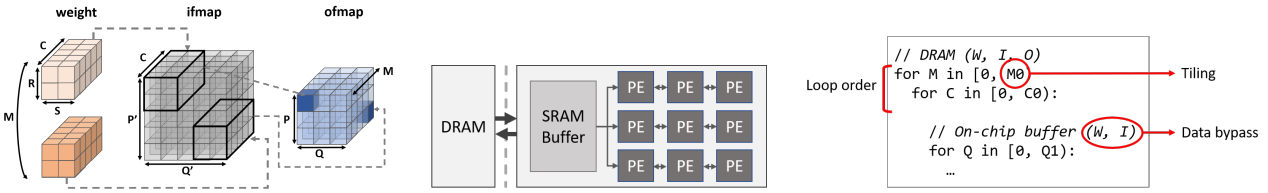
2 BACKGROUND

2.1 DNN Accelerator Design Space Exploration

DNNs [14, 25, 28, 36, 38, 41] are comprised of multiple layers. A multi-dimensional convolutional (CONV) layer is widely used for image and video processing applications. The computation of a 2-dimensional CONV layer (shown in Fig. 1a) involves taking a 3-dimensional $P' \times Q' \times C$ tensor called the input feature map (`ifmap`) and M 3-dimensional tensors called `weights` with a size of $R \times S \times C$, performing convolution operations to produce a tensor called output feature map (`ofmap`) with the size of $P \times Q \times M$. [1] Fully-connected layers that compute matrix multiplication can also be described in this form by setting P, Q, R and S to 1, and M and C to be the size of `ofmap` and `ifmap` vectors.

DNN accelerators are designed to exploit substantial data reuse within this multi-dimensional convolution and matrix multiplication computation. Given an architecture specification such as the number of processing elements (PEs) and on-chip buffer sizes, a designer aims to optimize performance and energy efficiency of an accelerator by figuring out the optimal *schedule* of the given DNN workload. A schedule describes how the computations and data movement are temporally and spatially mapped to hardware resources, and can be succinctly formulated using a nested for-loop called "loopnest" [32]. For example, in Fig. 1, we show an example architecture specification (Fig. 1b) and a sample loopnest schedule (Fig. 1c) corresponding to this architecture's memory hierarchy. The loopnest schedule in Fig. 1c describes the tiling strategy between memory levels and the multiplication-and-accumulate compute order. Note that a schedule is also referred to as a *mapping* in the literature [20, 32].

[1] $P = (P' - R + 2 \times \text{padding})/\text{stride} + 1$ and Q is derived similarly.



(a) DNN workload specification. Convolutions between `weight` and `ifmap` produce `ofmap`.

(b) An example DNN accelerator with the memory hierarchy.

(c) A sample loopnest (nested for-loops) with the specification for tiling, loop order, and data bypass.

Figure 1: Design space exploration of DNN accelerators.

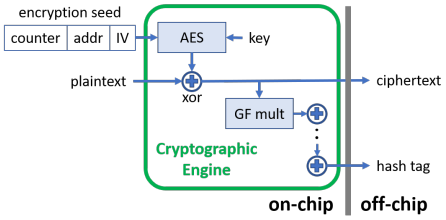


Figure 2: A cryptographic engine supporting AES-GCM, a widely used authenticated encryption protocol.

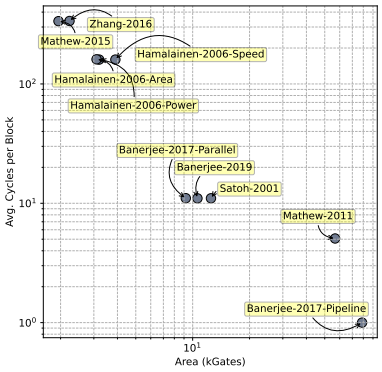


Figure 3: The tradeoff space for AES implementations.

Prior work acknowledges that the schedule search space for DNN accelerators is large, and efficiently searching for the optimal schedule presents a research challenge [4, 8, 15, 20, 32]. Several methodologies have been proposed. For example, Timeloop [32] used brute-force search over all possible loopnests, and supported approximate methods like random pruning to reduce the search time. CoSA [20], on the other hand, formulated the search problem as a constrained-optimization problem that can be solved using integer programming techniques. Furthermore, other classes of schedulers proposed to use machine learning driven approaches, such as [15].

The goal of this paper is to augment the design space exploration tools with the capability to take data encryption and authentication into account and find the optimal schedules for secure DNN accelerators.

2.2 Memory Encryption and Authentication

We consider both the confidentiality and integrity of data stored in the off-chip DRAM. A trusted execution environment (TEE) assumes that the on-chip structure are trusted and the off-chip memory is insecure. To ensure the confidentiality and integrity of data stored in the off-chip DRAM, cryptographic primitives, such as authenticated encryption, are often used.

An authenticated encryption scheme takes a plaintext, a secret key, and an encryption seed as inputs, and computes a ciphertext and a hash. Fig. 2 depicts the interface of a cryptographic engine that implements such a scheme with an explicit annotation on where each type of data is located. The hash is stored off-chip and is used to verify the integrity of the ciphertext. The encryption seed is composed of a counter, the address of the data, and a randomly generated initialization vector. The counter serves as a version number for the data and is incremented every time the accelerator generates a new version of the data. Since DNN accelerators use explicit data orchestration [34] and the accelerators have full knowledge of the version number, recent works [18, 19, 27] propose to track the counter using on-chip structures or the host CPU. Therefore, we assume the counters can be computed and accessing them does not incur complicated off-chip accesses.

All datatypes in a DNN (i.e., `weights`, `ifmaps`, and `ofmaps`) are in plaintext when they are stored and processed on-chip. When `ofmaps` or intermediate partial sums are generated and need to be written back to the DRAM, the cryptographic engine computes the ciphertext and hash corresponding to the data. When the data is fetched in the opposite direction, the accelerator retrieves the ciphertext data along with its associated hash from the DRAM, and feeds both into the cryptographic engine. The cryptographic engine validates the integrity of the ciphertext data against its hash and decrypts the data before supplying it to the on-chip components.

AES-GCM There are several standardized protocols for authenticated encryption, and among them, AES-GCM (Galois Counter Mode) has been widely used for its appealing characteristics in performance [37, 51]. As shown in Fig. 2, an AES-GCM block is primarily composed of an AES engine and a Galois-field multiplier. The encryption seed is fed into the AES engine to generate a one-time pad. Then, the one-time pad is XOR-ed with a plaintext to obtain a ciphertext, and vice versa. A hash is computed from a ciphertext using the Galois-field multiplication.

When performing design space explorations for secure DNN accelerators, we need to account for the overhead introduced by the cryptographic engine. There exists a variety of AES implementations with diverse performance and area characteristics. For example, Fig. 3 compares the AES hardware accelerator

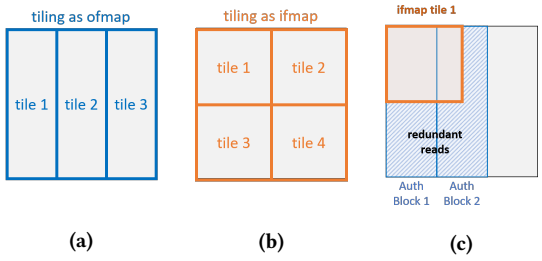


Figure 4: The same piece of data is used as `ofmap` of one layer (a) and as `ifmap` of the next layer (b) and the two layers use different tiling configurations. (c) shows that redundant reads are introduced when using the data as `ifmap` but assigning Authblock following `ofmap`'s tiling organization.

implementations published between 2001-2018 in circuits literature [2, 3, 12, 29, 42, 53]. It shows a clear trade-off between performance and area, where performance is measured by the average latency of encrypting/decrypting a 128-bit block (y-axis) and the area is counted by the number of equivalent gates to fairly compare among different technologies (x-axis). Our design space exploration tool, SecureLoop, can help select the appropriate cryptographic engine architecture to achieve an optimal performance/area trade-off.

3 MOTIVATION AND GOALS

We aim to develop a design space exploration tool for secure DNN accelerators, equipped with a search algorithm that identifies the optimal scheduling considering the unique properties of secure DNN accelerators. In this section, we point out that a cryptographic engine, which are often considered as a low-cost add-on to a predefined DNN accelerator in prior work [18, 19, 27], can pose significant overhead to different designs. Besides, we point out that authentication block assignments introduce a significant amount of complexity to our scheduling search space that our tool needs to navigate.

3.1 Overhead Due to Cryptographic Engines

Existing work on designing secure DNN accelerators [18, 19, 27] overlooks the fact that cryptographic engines can pose non-trivial overhead to the performance, energy, and area of the accelerator design and significantly shift the optimal design choices. As shown in Fig. 3, existing cryptographic engines do not achieve area-efficiency while attaining high performance at the same time. To make the point clearer, consider the DNN accelerators that target low-power and resource-constrained embedded platforms and IoT devices, such as Eyeriss [6] and other designs [13, 30]. To augment these accelerators with cryptographic engines to support a TEE, for example, we can use one AES-GCM engine that handles encryption and authentication for each datatype (i.e., `ifmap`, `ofmap`, and `weight`) as in [27]. When each AES-GCM engine is composed of a fully-pipelined AES engine and a single-cycle Galois-field multiplier [2], this configuration requires 416.7kGates in area, approximately 35% of the logic gates in Eyeriss [6], incurring extensive area overhead.

We note that prior work [18, 19, 27] only considered solutions for power-hungry accelerators, such as TPU [22], with large silicon area (e.g., $> 100\text{mm}^2$), and those design choices are not transferable to low-power and energy-efficient accelerators. Furthermore, the throughput of cryptographic engines has non-trivial impact on the loopnest scheduling. As cryptographic operations, such as encryption/decryption and authentication, accompany off-chip accesses, the supply of off-chip data to a DNN accelerator can be throttled by cryptographic engines if they have insufficient throughput. So far, we have shown that cryptographic engines complicate the design space of secure DNN accelerators. Our tool, SecureLoop, strives to perform a holistic assessment of the overhead due to cryptographic engines.

3.2 Authentication Block Assignment

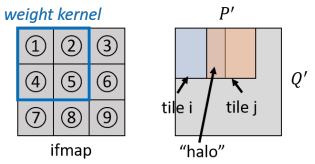
Authentication block assignment is a critical challenge for our design space exploration tool, as it extensively complicates the scheduling for secure DNN accelerators. Recall from Section 2.2, to perform memory authentication, a hash is computed for each block of off-chip data to verify its integrity. We call the unit of data that a hash is associated with an *authentication block*, or *AuthBlock* for short.

In prior work [18, 19], an authentication block is assigned using a strategy we refer to as "*tile-as-an-AuthBlock*". Specifically, in DNN accelerators, data is grouped into tiles and off-chip access is performed at the granularity of a tile. The size of the tile can be chosen to optimize for data reuses. The "tile-as-an-AuthBlock" strategy assigns authentication blocks to exactly match each datatype's tile organization.

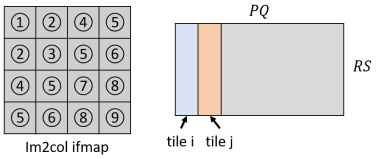
3.2.1 Cross-layer dependency. "Tile-as-an-AuthBlock", as a simple strategy, optimizes for minimizing the amount of hash reads for an individual DNN layer. However, it can incur unforeseen overhead due to *cross-layer dependency*. Cross-layer dependency arises from the characteristic of a DNN that the output feature map (denoted as `ofmap`) of one layer serves as the input feature map (denoted as `ifmap`) of the next layer. Fig. 4 provides an example to illustrate how such a dependency complicates the data traffic due to the AuthBlocks.

Consider a piece of data, when served as `ofmap`, the tiling strategy divides the data into 1×3 tiles. When served as `ifmap`, the tiling strategy divides the data into 2×2 tiles. We are running into a situation where we need to find an AuthBlock assignment strategy for the same piece of data that will be accessed by the accelerator with distinct patterns. If we follow the "tile-as-an-AuthBlock" strategy as in prior work, when assigning AuthBlock according to the `ofmap` tiles, we end up with a significant amount of redundant accesses when the data is served as `ifmap`. As shown in Fig. 4(c), when the accelerator fetches the first `ifmap` tile for DNN computation, it is forced to fetch the whole AuthBlock 1 and 2, doubling the off-chip traffic.

One workaround to reduce the redundant data accesses is to allow two different AuthBlock assignments for the same piece of data, which require a potentially high-cost "rehash" operation. Specifically, the AuthBlock assignment of the data was first optimized for `ofmap` access patterns (e.g., using "tile-as-an-AuthBlock"). Before the data is used as `ifmap`, the accelerator reads the data into the accelerator, fully decrypts the data, and re-assigns hashes based on a different AuthBlock organization that is optimized for `ifmap` access patterns. Rehashing introduces extra delays and off-chip traffic,



(a) Directly computing CONV can result in "halos" (overlaps) between tiles.



(b) Converting `ifmap` with `im2col` generates a larger matrix that has duplicated data, and tiles do not overlap.

Figure 5: Compare `ifmap` tiles for two different accelerators, one that directly supports CONV (a), and the other that computes matrix multiplications after converting CONV using `im2col` (b).

degrading the performance overall. Thus, to avoid rehashing, we aim to find the unified AuthBlock assignment considering different tiling strategies for one layer's `ofmap` and the next layer's `ifmap`.

3.2.2 Halos. Another problem that the "tile-as-an-AuthBlock" strategy faces is for convolution accelerators that directly perform CONV layers, instead of converting them to matrix multiplications using `im2col`. Fig. 5 compares how tiles are organized for the two different types of accelerators. Fig. 5(a) shows that, due to coarse-grained tiling, the accelerators dedicated for convolutions can have overlaps between tiles, especially in the `ifmap` datatype. We refer to the overlapping region as a "*halo*" throughout this paper. However, in Fig. 5(b), in the matrix multiplication case, each element exclusively belongs to one tile and there does not exists any overlap between tiles.

The existence of halos makes "tile-as-an-AuthBlock" an unappealing strategy. If we allow two AuthBlocks to share the overlapping data, we are forced to *duplicate* the halo data by encrypting and authenticating the data at least twice using different encryption seeds, which are composed of different counters, addresses, and initialization vectors. As a result, both the off-chip traffic and the memory footprint overhead are increased. Alternatively, not duplicating the halo data can result in large redundant reads if some AuthBlocks span across both the non-overlapping data and the halo data in one tile. In SecureLoop, we aim to search for the AuthBlock assignment to minimize the additional off-chip traffic caused by halos.

3.2.3 Goal of AuthBlock Assignment. To summarize, AuthBlock assignment poses a critical challenge in identifying the optimal scheduling for secure DNN accelerators, primarily for two reasons. First, the misalignment between AuthBlocks and data tiles, caused by cross-layer dependency or halos, leads to redundant data fetches for cryptographic authentication rather than DNN computation. Second, cross-layer dependency due to the AuthBlock assignment implies that the loopnest scheduling of two layers becomes fundamentally intertwined. There might be a loopnest schedule for one layer that is not optimal on its own, but results in better overall performance when it is considered together with its next layer.

Our design space exploration tool, SecureLoop, aims to search for the optimal AuthBlock assignment strategy to reduce off-chip traffic and maintain high performance. We consider the impacts of both the size and the orientation of the AuthBlocks and examine how they affect the additional off-chip traffic. In addition, we consider cross-layer dependency directly from the loopnest scheduling level, and search for schedules that optimize for global performance rather than a single-layer performance. In Section 5.1, we demonstrate that using an optimal AuthBlock assignment and performing the cross-layer optimization can provide 3-33% faster schedules and reduce the additional off-chip traffic from cryptographic operations by 37-94% compared to the "tile-as-an-AuthBlock" strategy.

4 SECURE ACCELERATOR SCHEDULING

We present SecureLoop, a design space exploration tool that is equipped with a scheduling search engine (Fig. 6) for secure DNN accelerators.

First, we introduce a simple model to estimate the performance and energy overhead of various cryptographic engines. The estimated cost is used to properly configure the architecture parameters, such as the off-chip bandwidth, of the existing loopnest scheduling algorithms. This approach is general enough to be compatible with a broad range of existing loopenst scheduling algorithms, such as Timeloop [32] and CoSA [20].

Second, we design a methodology to search for optimal authentication block assignment that takes both the size and the orientation of AuthBlocks into consideration. The key research challenge is that counting the amount of extra off-chip traffic caused by integrity verification via detailed simulation has scalability issues and cannot cope with a large search space. The approach that we take to address this scalability issue is to formulate the counting problem as a mathematical linear congruence problem and solve it efficiently.

Finally, we design a cross-layer fine-tuning stage to optimize both the scheduling and authentication block assignment strategy for cross-layer dependencies. The research challenge here is that the search space is amplified exponentially when we consider multiple layers together, especially for DNN workloads with a large number of layers, such as MobilenetV2 [41]. We use simulated annealing by heuristically defining neighboring loopnest configurations, and search for the final schedule.

4.1 A Model for Cryptographic Operations

We aim to identify the loopnest schedules for secure accelerators by leveraging the existing DNN loopnest schedulers. Recall that the difference between a secure DNN accelerator and a traditional accelerator is the extra cryptographic operations performed by the augmented cryptographic engine. The DNN loopnest schedulers have to be modified to account for those cryptographic operations. We adopt a simple and integrable solution that models the cryptographic operations as an additional constraint upon the off-chip DRAM bandwidth. Given that each piece of off-chip data access

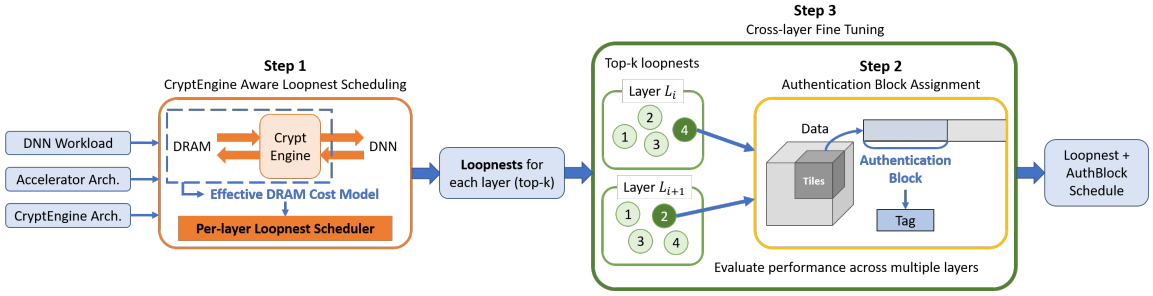
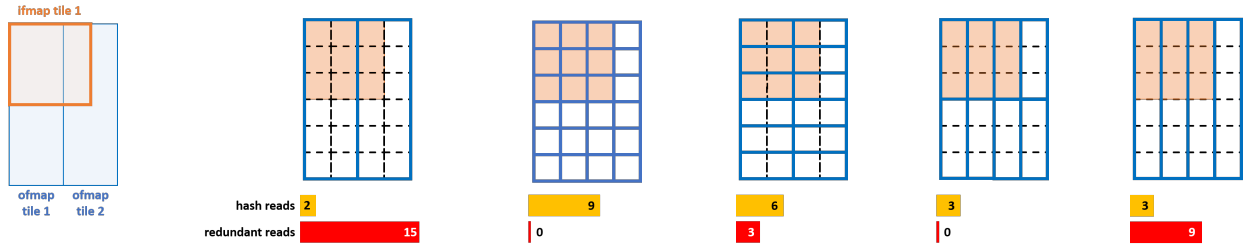


Figure 6: Overview of the scheduling search engine of SecureLoop.



(a) Cross-layer Dependency (b) tile-as-an-AuthBlock (c) horizontal, size: 1 (d) horizontal, size: 2 (e) vertical, size: 3 (f) vertical, size: 6

Figure 7: Examples of different AuthBlock assignments and their corresponding hash reads and redundant reads overhead. (a) reassembles the cross-layer dependency example discussed in Section 3.2. (b)-(f) describes 5 different authentication block assignment strategies. Each AuthBlock is marked with solid blue lines and the corresponding caption describes the AuthBlock orientation and size.

needs to go through both the DRAM interface and the cryptographic engine, the slower component among the two limits the off-chip bandwidth. Thus, we derive the effective off-chip bandwidth of a secure accelerator by taking the minimum of the memory bandwidth and the cryptographic engine bandwidth. This effective bandwidth replaces the original memory bandwidth for loopnest scheduling purposes. Such an approach is highly compatible with loopnest search tools whose internals may vary significantly. Besides, this approach is in line with the assumption common among existing search tools, that is, different hardware components on the DNN accelerator are appropriately pipelined with negligible pipelining overhead (e.g., using techniques such as double-buffering or buffets [34]).

4.2 Mathematical Formulation for Authentication Block Assignment

In the second step of our scheduler, we aim to determine an optimal authentication block assignment strategy that can minimize the additional off-chip traffic caused by data authentication, and thus minimize the extra overall performance overhead. This step requires performing an exhaustive search over all feasible AuthBlock sizes and orientations for each layer and datatype (i.e., `weight`, `ifmap`, and `ofmap`). Such a search poses a serious scalability issue, which we address with a mathematical formulation of the problem.

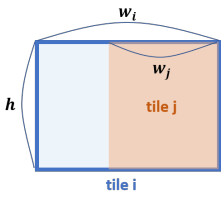
The Search Space We start by describing what the search space for authentication block assignment looks like.

Given the nature of the memory authentication operation, it introduces additional off-chip memory accesses, classified into two categories. First, extra accesses to fetch the hashes. Second, extra accesses to fetch the data that is not needed for the actual DNN computation, but is needed for integrity verification because it lies within the same authentication block as the data used by the accelerator. We distinguish the two types of overhead as *hash reads* and *redundant reads* respectively.

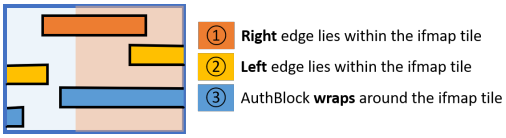
There exists a non-trivial search space for authentication block assignment, because both the size and orientation of the authentication block affect the off-chip traffic overhead. We provide examples in Fig. 7 to illustrate the search space and highlight the trade-off between hash reads and redundant reads with different AuthBlock assignments. In each figure, we highlight the first `ifmap` tile in orange, mark each authentication block with solid blue lines, and we list the hash reads and redundant reads at the bottom of each assignment.

In Fig. 7(a) and (b), the example reassembles the cross-layer dependency case described in Section 3.2, where the AuthBlock is assigned according to the `ofmap` tiling. Since there are two AuthBlocks in total, the hash reads overhead is low. However, large redundant reads are incurred as all data belonging to AuthBlock 1 and 2 has to be fetched when accessing the first tile.

Fig. 7(c) and (d) compares two cases of using a horizontal AuthBlock with varied size. In (c), it is an extreme case where the AuthBlock size is 1, meaning each element is assigned with a hash, resulting in high hash reads overhead with zero redundant reads. In (d), when we increase the AuthBlock size from 1 to 2, the hash reads are reduced by half, but we start to have redundant reads because some of the AuthBlocks span across the boundary of the first tile. These two cases clearly demonstrate the impact of the size



(a) An example of a mismatch between the tile$_i$ and the tile$_j$.



(b) Three conditions for an AuthBlock to lie in the intersection of the tile$_i$ and the tile$_j$ from the above example.

Figure 8: Mathematical formulation of counting redundant reads for a given AuthBlock assignment.

of AuthBlocks. When the AuthBlock size increases, the hash reads decrease, but the redundant reads can increase.

To further complicate the space, the orientation of the AuthBlock also matters. Fig. 7(e) shows vertical AuthBlocks with a size of 3. This strategy happens to be an ideal strategy, because every AuthBlock resides exactly within the first `ifmap` tile, leading to no redundant reads. Meanwhile, since the AuthBlock size is 3, it has 1/3 of the hash reads overhead compared to the horizontal size-1 strategy shown in Fig. 7(c). However, if we increase the size of the vertical AuthBlock to 6 in Fig. 7(f), the amount of redundant reads increases.

In summary, both the orientation and size of an AuthBlock affect the off-chip traffic overhead. We perform an exhaustive search to identify the optimal AuthBlock assignment, and the search has to be efficient.

Mathematical Formulation We now describe our mathematical formulation of the authentication block search process. This formulation will enable us to solve the problem analytically. Our formulation can be applied to the two cases where redundant reads occur: 1) cross-layer dependency, and 2) halos (Section 3.2).

In both cases, we are given a piece of data, its tile organizations where several tiles overlap (e.g., overlaps between the `ofmap` tiles and the `ifmap` tiles for cross-layer dependency, or overlaps between tiles among the `ifmap` tiles for halos). We are asked to calculate the number of redundant reads and hash reads for each AuthBlock assignment. Since each time an AuthBlock is accessed, all the elements in that AuthBlock need to be fetched together, we reduce the problem of calculating redundant reads to counting *the number of AuthBlocks that overlap with each tile.*

We convert the above problem into a linear congruence problem as follows using an example in Fig. 8. The example shows a 2D tensor with two overlapping tiles, called tile$_i$ and tile$_j$. For illustration purposes, assuming the two tiles have the same height, h, and different widths, denoted as w_i and w_j. Let's consider the case when we assign horizontal AuthBlocks to fully cover tile$_i$, so that no redundant access is needed when accessing tile$_i$ (if tile$_i$ is the `ofmap` tile, this will be a natural scenario as hashes will be computed as the `ofmap` is generated). These AuthBlocks may not fully align with the boundary of tile$_j$, and thus we need to handle the case when the AuthBlocks crosses the boundary of tile$_j$. The AuthBlocks can overlap with tile$_j$ in three conditions, shown in Fig. 8(b).

We denote an AuthBlock using the (x, y) coordinates of its left and right edges. Specifically, an AuthBlock has its left edge labeled as (L_x, L_y) and it right edge labeled as (R_x, R_y). Then the following mathematical conditions can be used to precisely capture the three conditions. The first two scenarios are straightforward, where either the right side or the left side of the AuthBlock overlaps with tile$_j$:

$$w_i - w_j \leq R_x < w_i \tag{1}$$

$$w_i - w_j \leq L_x < w_i \tag{2}$$

The third scenario describes an AuthBlock wraping around tile$_j$ with both of its left and right edges located in tile$_i$ (assuming the tile size is lesser than w_i).

$$L_x < w_i - w_j \quad \wedge \quad R_x < w_i - w_j \quad \wedge \quad R_x < L_x \tag{3}$$

With the above formulation, we set out to efficiently calculate the number of AuthBlocks that can satisfy one out of the three formulas above. Assuming an AuthBlock configuration with a height of 1 and a varied width denoted as u, the L_x, R_x for the k-th AuthBlock is derived as $L_x^k = (u \times k) \bmod w_i$ and $R_x^k = (u \times k + (u-1)) \bmod w_i$, and can be plugged into the three formulas. Then, solving the inequalities for u and k, and converting the inequalities into the linear congruence equation by listing all possible values satisfying the inequalities, we obtain the following linear congruence problem:

$$u \times k \equiv \min(w_i - w_j - u + 1, 0), \cdots, w_i - 1 \pmod{w_i} \tag{4}$$

The formula above uses modular arithmetic, where $a \equiv b \pmod{c}$ means that the remainders of a and b divided by c are equal. We then end up counting how many occurrences of k ($0 \leq k < \lceil \frac{h \times w_i}{u} \rceil$) satisfy Eq. (4). This linear congruence problem can be efficiently solved using the extended Euclidean algorithm that finds the greatest common denominator. We can use this algorithm to find all ks in a log-linear time.

The above example demonstrates horizontal AuthBlock assignment and assumes 2D tiles with the same height. However, the discussed methodology above is general enough to be applicable to vertical AuthBlock assignments and for higher-dimensional tiles with arbitrary overlapping patterns. To generalize the problem, consider a n-dimensional tile. We search for AuthBlocks with $n-1$ of the dimensions set to 1 and the remaining dimension u to be varied. In essence, we are flattening an n-dimensional tensor to a 1-d vector and slicing it. Therefore, our computation complexity increases linearly as the possible value of the width of AuthBlocks, i.e., u, whose maximum value is capped by the number of elements in a tile, regardless of the dimension size.

Example of Analysis Results In Fig. 9, we visualize the search space of AuthBlock assignment. The example follows the setup in Fig. 8 by setting $h = 30$, $w_i = 30$, and $w_j = 20$. We then sweep the AuthBlock size u from 1 to 30 for the horizontal orientation (note that $u > 30$ will result in the same redundant reads as the "tile-as-an-AuthBlock"), and from 1 to 900 for the vertical orientation, where the upper bound means using the full tile as an AuthBlock, to see how these variations affect the overall off-chip traffic when accessing the misaligned tile.

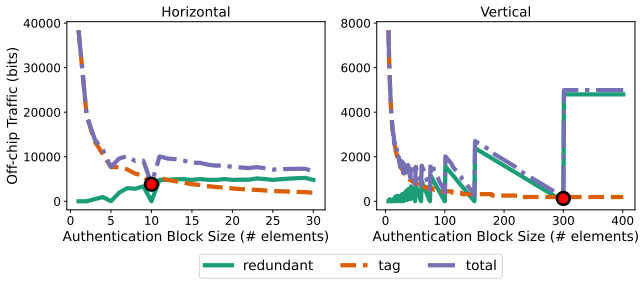


Figure 9: The amount of off-chip traffic incurred for accessing tile$_j$ in Fig. 8 when varying the AuthBlock orientation and size.

In both figures, we observe an inversely proportional relationship between the AuthBlock size and the amount of hash reads. When we use horizontal AuthBlocks, we observe that the overall trend between the redundant reads and the AuthBlock size is a positive linear relationship, but there exist several distinguishable local valleys. We observe the optimal assignment choice is to set $u = 10$, which hits a local minimal value of the redundant reads, and meanwhile incurs a moderate level of hash reads overhead. When using vertical AuthBlocks, the trade-off space is rather irregular. Since the two tiles in Fig. 8 have the same height, we periodically observe zero redundant reads whenever the AuthBlock size is a factor of $h \times (w_i - w_j) = 300$. Using an exhaustive search, we identify the optimal AuthBlock size is 300.

4.3 Efficient Cross-layer Fine Tuning

Cross-layer dependency interweaves the loopnest scheduling of two consecutive layers. So far, we derived the loopnest schedules with the best individual layer performance in the first step of our scheduler, and identified the optimal AuthBlock assignment based on those loopnest schedules. However, the obtained schedule may not be the global optimum considering the dependency. To account for cross-layer dependency from the loopnest scheduling level, we introduce the third step in our scheduler that fine tunes the final schedule.

Challenges Traditional schedulers for DNN accelerators usually search for the optimal scheduling for each layer independently, and they cannot be easily adapted to consider the influence of cryptographic operations.

The search space for loopnest scheduling increases exponentially with the number of layers we have to search jointly. For brute-force search algorithms [4, 8, 32], computational complexity imposed by cross-layer dependency can become prohibitive for deep models [14, 41]. Other algorithms, such as optimization-based techniques [20], is unlikely to be applicable due to the AuthBlock assignment, as the mathematical formulation for the AuthBlock assignment (Section 4.2) cannot be easily reduced to a closed-form. Moreover, it cannot be guaranteed to result in convex or linear constraints on objective functions. There is only limited work on jointly searching the loopnest schedules for multiple layers, such as in the context of fused-layer processing [43]. We consider those efforts to be promising yet orthogonal to our work.

Search Using Simulated Annealing We propose to use simulated annealing, a metaheuristic algorithm. Simulated annealing is a probabilistic method for solving an optimization problem over a large search space. It iteratively searches for the *neighbors* of the current state and probabilistically decides whether to move on to the new state. This probability is determined by the difference in the costs of the current state and the new state, and a parameter called *temperature*. The temperature is gradually decreased, such that suboptimal yet diverse states can be explored in the earlier iterations, while the best solutions can be exploited in the later iterations.

Algorithm 1 Pseudocode for step 3: simulated annealing

1: $L_1, ...L_n \leftarrow L_1^\circ, ..., L_n^\circ$
2: `cost` $\leftarrow$ `PerfModel`$(L_1, ..., L_n)$
3: $t \leftarrow T_{\text{init}}$ `{initialize temperature}`
4: **for** $n \leftarrow 1, ..., N$ **do**
5: $i \leftarrow$ `random`$(1, ..., n)$
6: $L_i' \leftarrow$ `GetNeighbor`$(\mathcal{L}_i)$
7: `cost'` $\leftarrow$ `PerfModel`$(L_1, ..., L_i', ..., L_n)$
8: `cost_diff` = `cost` − `cost'`
9: **if** $\exp \frac{\texttt{cost\_diff}}{t} >$ `random.uniform`$(0, 1)$ **then**
10: $L_i \leftarrow L_i'$ `{probabilistic accept the new schedule}`
11: `cost` $\leftarrow$ `cost'`
12: **end if**
13: $t \leftarrow$ `GetTemperature`$(t, n, T_{\text{init}}, T_{\text{final}})$
14: **end for**

Algorithm 1 describes our adaptation of simulated annealing algorithm for cross-layer fine tuning. We denote L_i° as the optimal loopnest schedule of the i-th layer found from the first step without considering cross-layer dependency. Our algorithm attempts to identify a set of loopnest schedules $(L_1, \cdots, L_n)$ that results in better performance compared to $(L_1^\circ, \cdots, L_n^\circ)$ when n layers are considered altogether.

The algorithm starts by initializing the current set of loopnest schedules as $(L_1^\circ, \cdots, L_n^\circ)$ and calculates its cost using the performance model and the optimal AuthBlock assignment (lines 1-2). Then, for each iteration, the algorithm randomly selects one layer i and a neighboring schedule L_i' (line 6) for that layer. Observe that the key component of this algorithm is a heuristic involved in proposing a neighbor (the `GetNeighbor` function). There is no natural metric for measuring the similarity between two loopnest schedules. Our scheme uses the per-layer performance as the similarity metric. Specifically, we obtain top-k best loopnest schedules per layer from the first stage loopnest scheduler, and the `GetNeighbor` function randomly samples among these k different loopnest schedules to get a neighbor. When searching among k possible schedules for each layer, the search space has k^n distinct combinations to be explored in a limited number of simulated annealing iterations. The new schedule that replaces the loopnest schedule of the i-th layer with L_i' is probabilistically accepted (lines 9-12), and the temperature is decreased linearly (line 13).

Impact of Search Parameters We examine how the search parameters affect the search result.We demonstrate the performance improvement when the simulated annealing method is used with different values of k (the number of top schedules per layer to form the neighbor set) in Fig. 10. The numbers are for an architecture derived from Eyeriss [6] with a cryptographic engine with

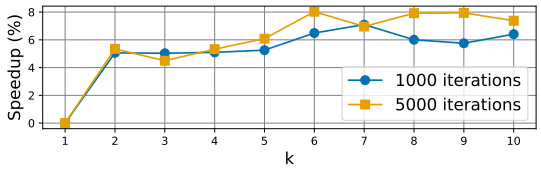


Figure 10: Improvement in latency (speedup) when using simulated annealing for different values of k, compared to when only the top-1 loopnest schedule for each layer.

an energy-efficient AES-GCM implementation from [2] (detailed specifications in Table 2) running a MobilenetV2 [41] workload.

Increasing k from 1 to 2 can improve the overall performance about 5%. However, further increasing k results in a slower performance improvement, and the improvement stalls around the point when $k = 6$. Considering that a larger k does not always result in better speedup but can substantially increase the search space size, we set $k = 6$ for subsequent experiments. Also, the number of iterations directly affects the search time, and we use 1000 iterations as a default setup to trade-off the quality of results and the search time.

Handling Post-processing Operations Our cross-layer fine tuning needs to consider post-processing operations between consecutive layers, such as Batch Normalization [21], activation functions, and pooling operations. There exist two cases.

First, some post-processing operations can be performed on-the-fly while the `ofmap` is being generated, thus can be considered as part of that layer. We consider Batch Normalization, ReLU activation, and adding zero pads around the feature map to fall into this case, as they are simple operations. We handle such operations using the cross-layer fine-tuning approach discussed above.

Meanwhile, some other post-processing operations cannot be performed on-the-fly together with in-layer computation, such as pooling operations and operations to combine several feature maps for residual connections. The existence of these operations requires a separate computation step, and inevitably triggers rehashing. Thus, the cross-layer dependency problem for AuthBlock assignment does not exist. As such, given a full DNN, we divide them into multiple segments based on the existence of such post-processing operations and apply fine tuning within each segment.

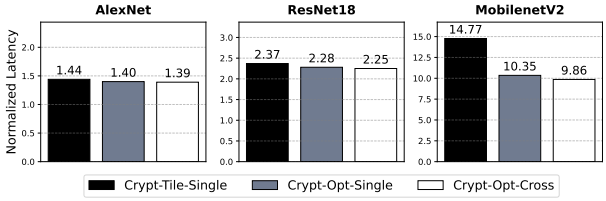
5 EVALUATION

We first present the effect of scheduling algorithms on the performance of a secure accelerator in Section 5.1. Then, we show the performance of diverse secure DNN accelerator designs, that vary in the choice of cryptographic engines, the number of PEs, and the size of the on-chip global buffer (Section 5.2). From these experiments, we show the area-performance trade-off for secure accelerators, and provide insights on the Pareto optimal design points (Section 5.3).

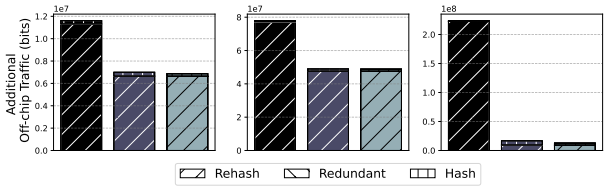
Base Architecture Configuration We consider diverse DNN accelerator designs in the following experiments derived from a base configuration. As the base configuration, we use a spatial DNN accelerator with multiple processing elements (PEs), where each PE has an ALU and a small local memory, operating in parallel and organized as a 2-dimensional array of shape $X \times Y$. The base configuration has an on-chip SRAM buffer, and the data movement can be described by its dataflow. We set the base configuration to use the row-stationary dataflow from [6], 14×12 PEs, and 131kB on-chip global buffer.

Table 1: Summary of different scheduling algorithms.

Scheduling Algorithm	Loopnest Scheduler	AuthBlock	Cross-layer Fine Tune?
Crypt-Tile-Single	Crypt-Aware	Tile-as-an-AuthBlock	N
Crypt-Opt-Single	Crypt-Aware	Optimal-AuthBlock	N
Crypt-Opt-Cross	Crypt-Aware	Optimal-AuthBlock	Fine Tune



(a) Performance overhead using different scheduling algorithms, measured by the number of cycles normalized to the unsecure baseline accelerator.



(b) The additional off-chip traffic along with its breakdown into hash reads, redundant reads, and rehashing traffic for different scheduling algorithms.

Figure 11: Impacts of scheduling algorithms on performance and off-chip traffic.

5.1 Effect of the Scheduling Algorithm

We examine the effect of different scheduling algorithms (Table 1) on the performance of secure accelerators. As a baseline scheduling algorithm, we consider *Crypt-Tile-Single*, which indicates that it uses the Timeloop [32] with the effective bandwidth and energy for the off-chip access reflecting the cryptographic operations, the "tile-as-an-AuthBlock" assignment strategy, and does not consider cross-layer dependency. We note that supplying the proper bandwidth and energy parameters to Timeloop is crucial to prevent suboptimal loopnest schedules degrading the baseline performance especially when the cryptographic engine has low throughput. We then add the second and third step one by one, with the most optimized version denoted as *Crypt-Opt-Cross* with both the optimal AuthBlock assignment and the cross-layer search enabled. The first step of our scheduler is implemented upon Timeloop, but with an extension to support top-k loopnests searching and modifications to the effective energy and bandwidth for the off-chip accesses. The second and third steps of our scheduler are implemented as independent modules that accept the top-k loopnest schedules for each layer as inputs, and return the final loopnest schedule and optimal AuthBlock assignments.

We evaluate our scheduling algorithms on the DNN accelerator design with the base configuration we described before. The secure accelerator uses an area-efficient parallel AES-GCM implementation [2, 3] as its cryptographic engine (one per each datatype). For the off-chip DRAM access, we assume LPDDR4 with the throughput of 64B/cycle. Accelergy [49] is used to estimate energy and area of each component on the DNN accelerator, assuming 40/45nm technology it supports.

Fig. 11(a) shows the slowdown in secure accelerators, i.e., the number of cycles to process a workload normalized to that of baseline (unsecure) accelerators. Fig. 11(b) shows the additional off-chip traffic incurred by cryptographic operations for each scheduling algorithm. We examine three workloads with varying number of layers and characteristics: AlexNet [25] , ResNet18 [14] , and MobilenetV2 [41] . These workloads are mainly convolutional, and note that we only consider first 5 layers of AlexNet that are convolutional.

First, our optimal AuthBlock assignment strategy reduces the additional off-chip traffic across all three DNN workloads compared to the "tile-as-an-AuthBlock" assignment. The benefit comes from two factors: 1) rehashing operations are not necessary between dependent layers as the AuthBlocks are assigned by considering the mismatches between their tiling strategies, and 2) both redundant reads and hash reads are minimized without having to rehash or duplicate some data. Also, this step reduces the slowdown by up to 29.9% compared to *Crypt-Tile-Single* as well. These two factors affect deeper workloads more significantly, and the benefit of the AuthBlock assignment is most visible in MobilenetV2.

Second, cross-layer fine tuning of our scheduling primarily improves the performance for a deep workload like MobilenetV2 with additional 3.3% improvement on top of *Crypt-Opt-Single*. Simulated annealing involves stochasticity when choosing a neighbor, and the performance gain from this step can vary due to randomness. From 5 independent runs for simulated annealing, we observe that the slowdown for MobilenetV2 with *Crypt-Opt-Cross* can vary from 9.76 to 9.99 with the standard deviation of 0.08, and Fig. 11(a) reports the mean value. This step does not significantly affect the performance on a more shallower workload like AlexNet, where the opportunity for cross-layer optimization is limited. Nevertheless, it is worthy to note that this step reduces the additional off-chip traffic due to redundant reads and hash reads (excluding rehashing-related traffic) by 32.6% and 16.0% even for AlexNet and ResNet18. Overall, our scheduler results in a schedule that is up to 33.2% faster and 50.2% better in EDP compared to *Crypt-Tile-Single*.

Roofline Model We can also use the roofline model [48] to intuitively reason about the impact of scheduling algorithms (Fig. 12). In the left of Fig. 12, the roofline model describes the performance (y-axis) of each DNN workload, as a function of the computational intensity (x-axis). The computational intensity is measured by the number of operations (e.g., multiplication and addition) per byte of DRAM traffic, and performance is measured by the number of operations per second, assuming a 100MHz clock. There are two solid lines illustrating the maximum possible performance: the horizontal solid line is determined by the number of PEs that can operate in parallel, and the slanted solid line represents the performance limited by the off-chip memory bandwidth. The dotted slant line is based on the effective off-chip bandwidth of a secure DNN accelerator constrained by its cryptographic engine, assuming a single parallel AES-GCM engine processes every off-chip data transfer (in actual designs, each datatype has its own dedicated cryptographic engine, and the performance can be higher than this effective line). We can observe that the workloads were in the compute-bounded region for the unsecure baseline accelerator, but throttling from the cryptographic engine pushes the workloads to be in the effectively memory-bounded region in secure accelerators. The right part of Fig. 12 zooms in to show the different scheduling algorithms for the MobilenetV2 workload, and shows that each step in our scheduler improves the performance by finding schedules with higher computational intensity.

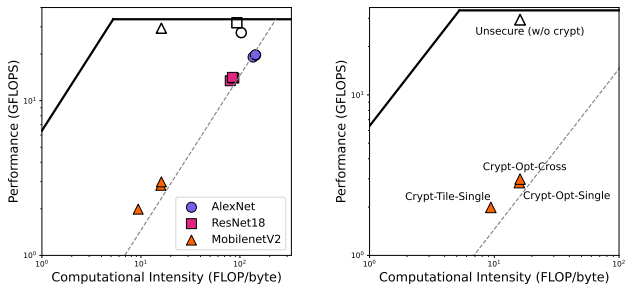


Figure 12: Left: Roofline model for accelerators using different scheduling algorithms. White markers represent the unsecure baseline, and colored markers represent secure accelerators. Right: Roofline model zoomed-in to show different scheduling algorithms for the MobilenetV2 workload.

5.2 Impacts on Architecture Configurations

We evaluate the impact of using different architecture configurations and cryptographic engines using SecureLoop.

Cryptographic Engine Configurations We evaluate the impact of different cryptographic engine configurations, varying in their AES-GCM engine architecture and counts, on the area overhead and the performance. We use three different AES-GCM engine implementations, summarized in Table 2. These designs have distinct characteristics in the area-throughput trade-off, with the fully-pipelined design supporting high throughput but large area overhead, whereas the serial design has low area overhead and low throughput. The parallel design is in between two other designs, with medium throughput and area overhead. The area of AES-GCM engines is normalized to 40nm technology using the equivalent number of gates [2, 53].

We use the same accelerator architecture as in Section 5.1 and use the *Crypt-Opt-Cross* scheduling algorithm. Fig. 13 compares the slowdown over the unsecure baseline design for each workload and the area overhead for each configuration. We find that similar performance can be obtained by configurations with very different area overhead. For example, the configuration with 30× serial AES-GCM engines has similar performance to the one with 1× parallel AES-GCM engine, although they have 10× difference in the area overhead. Thus, the scalability of area-efficient yet low-throughput AES-GCM engines can be problematic for DNN accelerators, and

Table 2: Specifications of AES and Galois-field multiplier (GFMult) used to construct an AES-GCM engine.

Architecture	AES			GFMult		
	Cycle	Area (kGates)	Energy (pJ)	Cycle	Area (kGates)	Energy (pJ)
Pipelined	1	78.8	165.1	1	60.1	57.7
Parallel	11	9.2	194.6	8	9.7	82.4
Serial	336	3.0	768	128	3.3	345.6

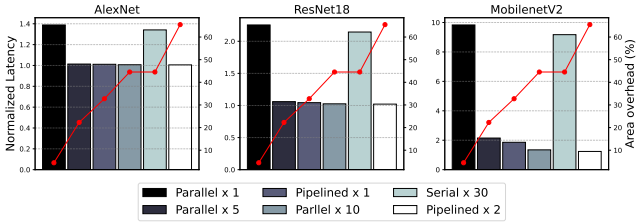


Figure 13: Slowdown over the unsecure baseline design and the area overhead of secure accelerators varying in their cryptographic engine configurations.

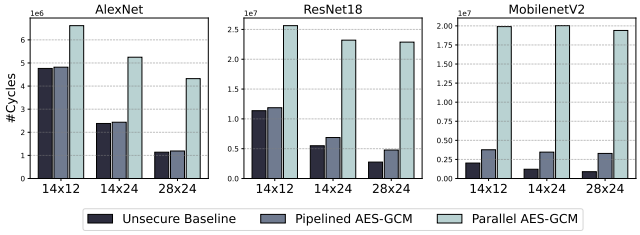


Figure 14: Latency for secure accelerator designs varying in their number of PEs.

often using a moderate number of higher-throughput AES-GCM engines is a better design choice.

Scaling the Number of Processing Elements We examine accelerator designs varying the number of PEs in the base configuration. We consider two cryptographic engine configurations, 1× pipelined AES-GCM engine and 1× parallel AES-GCM engine. Fig. 14 shows the evaluation result for different PE organizations 14×12, 14×24, and 28×24. The number of PEs determines the maximum possible performance of the accelerator if the memory bandwidth is sufficient, and this trend is well manifested for the unsecure baseline accelerators (the latency decreases almost by half as the number of PEs is doubled). However, since secure accelerators can be effectively bounded by the supply of decrypted data, the benefit of increasing the PE array size is not apparent for the design with a parallel AES-GCM engine. Thus, the performance of secure accelerators cannot be improved by more PEs unless the cryptographic engine throughput is also increased.

Scaling the Size of On-chip Buffer The size of the on-chip SRAM buffer limits the maximum tile size for the `ifmap` and `ofmap` for the row-stationary dataflow architecture we used. In Fig. 15, we examine the effect of different buffer sizes (131kB, 32kB, and 16kB) on secure accelerators while other design paramters are fixed. As we scale down the buffer size, the size of tiles between the off-chip and the on-chip buffer decreases, often resulting in larger off-chip traffic. For the unsecure baseline accelerators, larger off-chip traffic is not problematic because they have sufficient off-chip memory bandwidth. However, it can further throttle the secure accelerators

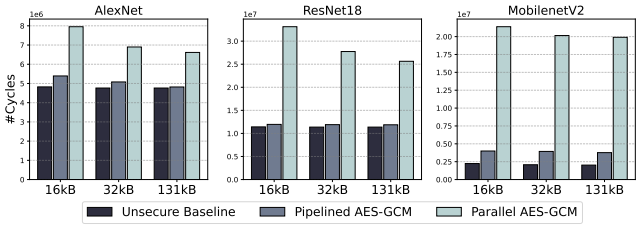


Figure 15: Latency of designs varying in the size of on-chip SRAM buffer.

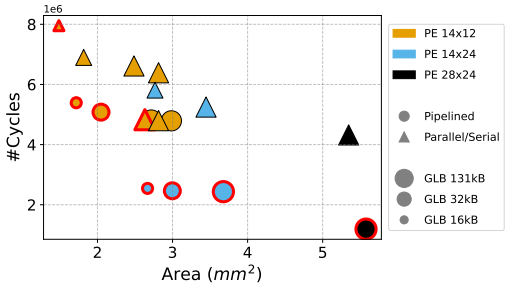


Figure 16: The area vs. performance trade-off of secure accelerator designs. Points highlighted with red edges indicate the Pareto front of this trade-off curve.

with limited encryption/decryption bandwidth, thus leading to longer latency for small buffer sizes.

Different DRAM Technologies An off-chip DRAM with higher bandwidth does not necessarily improve the performance of secure accelerators, as the effective off-chip bandwidth is limited by the cryptographic engine. However, the energy for the off-chip access is directly affected by the DRAM technology. To illustrate these two points, we experiment with three different DRAM configurations: LPDDR4 with 64B/cycle throughput, LPDDR4 with 128B/cycle throughput, and HBM2 with 64B/cycle throughput. For the AlexNet workload, we observe that the DRAM bandwidth does not affect the latency and energy of secure accelerators. HBM2 has lower energy per access compared to LPDDR4, and the energy for both the unsecured baseline and the secure accelerators decreases compared to LPDDR4, while the latency is not affected.

Impact of TEE Entry/Exit Entering a TEE and exiting from it can affect the performance when the full system is considered end-to-end. Previous works that examined the end-to-end overhead of supporting a TEE for accelerators [27] showed that the initial transfer of DNN weights to the accelerator context is the major source of latency for the entry. We note that this transfer latency might not vary significantly across different accelerator architecture, as the transfer is determined by the model parameter size and the host CPU. Furthermore, when an accelerator is serving multiple inference requests using the same DNN, this initial transfer cost of model parameters can be negligible compared to the overall execution time. Thus, we expect that TEE entries/exits do not significantly affect the optimal design of secure accelerators.

5.3 Area vs. Performance Trade-off

Finally, we plot the area vs. latency (for the AlexNet workload) trade-off curve for several designs we have discussed so far in Fig. 16. We also derive the Pareto front of this trade-off curve, and observe characteristics of the optimal and suboptimal points. First, the designs with a small on-chip buffer size but with a high throughput cryptographic engine (i.e., pipelined AES-GCM engines) are often optimal. As we observed in Fig. 15, performance is not degraded much if the cryptographic engine provides sufficient throughput even if we scale down the buffer size. Thus, dedicating more area to the cryptographic engine by reducing the on-chip buffer size can provide a good trade-off.

Besides, the designs with larger PE array sizes (e.g., 14×24 or more) but with a low throughput cryptographic engine result in suboptimal points. This observation agrees with Fig. 14 that the benefit of having higher parallelism cannot be achieved when cryptographic engines are the bottleneck.

6 RELATED WORK

We now discuss related work on supporting TEEs on general-purpose processors and DNN accelerators. We also cover alternative approaches to secure DNN computation.

Supporting a TEE for CPUs and GPUs Many optimizations have been proposed to reduce the overhead of supporting a TEE in general-purpose processors, such as CPUs and GPUs. Several works proposed techniques to reduce the overhead of traversing a Merkle tree in CPUs [9, 37, 51]. Different counter formats were proposed to allow a more compact Merkle tree [39, 45]. Recent works extended a TEE for GPUs and accelerators with a trusted I/O between a host CPU and these accelerators [1, 47].

Tree-less Verification for DNN Accelerators Recent works showed that the counters do not have to be stored, and instead can be *calculated* from the computation pattern of DNN accelerators [18, 19, 27], removing the necessity of a Merkle tree. [18, 19] tracked the counters by a MCU unit on the accelerator, and [27] proposed an external host processor to supply the counters. [18, 19] further proposed "tile-as-an-AuthBlock", and we considered this strategy as the baseline in our evaluation

Other Techniques for Secure Machine Learning [46] showed that a DNN computation can be delegated to an untrusted accelerator from the secure host CPU using a verifiable outsourcing algorithm. Homomorphic encryption that performs computations over the encrypted domain provides privacy to both the user input and the model, and several works proposed techniques to mitigate its overhead [23, 24, 35, 40]. Recent work explored hardware acceleration to support differential privacy as well [33]. Finally, [50] presented a method for securing computations over off-chip near data processing accelerators.

7 CONCLUSION

This work presents a framework for systematic design space exploration of secure DNN accelerators supporting a TEE for privacy and integrity. We present SecureLoop, that is equipped with a scheduling search engine capable of 1) cryptographic engine aware loopnest scheduling, enabled by a simple performance and energy modeling of a cryptographic engine, 2) the optimal AuthBlock assignment navigating a complex search space dependent on both the size and orientation of AuthBlocks, and 3) cross-layer fine tuning using simulated annealing. Using our framework, we show the impact of design parameters on the performance of secure accelerators, and provide design insights for secure accelerators from the area vs. performance trade-off.

ACKNOWLEDGMENTS

This work was funded in part by Samsung Electronics, Korea Foundation for Advanced Studies, and NSF PPoSS. We thank the anonymous reviewers for all their valuable comments during the review process.

REFERENCES

[1] 2022. *Software Enabling for Intel® TDX in Support of TEE-I/O.* Technical Report. Intel Corporation.
[2] Utsav Banerjee. 2017. *Energy-efficient protocols and hardware architectures for transport layer security.* Master's thesis. Massachusetts Institute of Technology.
[3] Utsav Banerjee, Andrew Wright, Chiraag Juvekar, Madeleine Waller, Arvind, and Anantha P. Chandrakasan. 2019. An Energy-Efficient Reconfigurable DTLS Cryptographic Engine for Securing Internet-of-Things Applications. *IEEE Journal of Solid-State Circuits* 54, 8 (2019), 2339–2352. https://doi.org/10.1109/JSSC.2019.2915203
[4] Prasanth Chatarasi, Hyoukjun Kwon, Angshuman Parashar, Michael Pellauer, Tushar Krishna, and Vivek Sarkar. 2021. Marvel: A Data-Centric Approach for Mapping Deep Learning Operators on Spatial Accelerators. *ACM Trans. Archit. Code Optim.* 19, 1, Article 6 (dec 2021), 26 pages. https://doi.org/10.1145/3485137
[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 269–284. https://doi.org/10.1145/2541940.2541967
[6] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379. https://doi.org/10.1109/ISCA.2016.40
[7] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).
[8] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. 2019. DMazeRunner: Executing Perfectly Nested Loops on Dataflow Accelerators. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 70 (oct 2019), 27 pages. https://doi.org/10.1145/3358198
[9] B. Gassend, G.E. Suh, D. Clarke, M. van Dijk, and S. Devadas. 2003. Caches and hash trees for efficient memory integrity verification. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.* 295–306. https://doi.org/10.1109/HPCA.2003.1183547
[10] Shay Gueron. 2016. Memory Encryption for General-Purpose Processors. *IEEE Security and Privacy* 14, 6 (nov 2016), 54–62. https://doi.org/10.1109/MSP.2016.124
[11] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. 2009. Lest We Remember: Cold-Boot Attacks on Encryption Keys. *Commun. ACM* 52, 5 (may 2009), 91–98. https://doi.org/10.1145/1506409.1506429
[12] P. Hamalainen, T. Alho, M. Hannikainen, and T.D. Hamalainen. 2006. Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In *9th EUROMICRO Conference on Digital System Design (DSD'06)*. 577–583. https://doi.org/10.1109/DSD.2006.40
[13] 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. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA '16)*. IEEE Press, 243–254. https://doi.org/10.1109/ISCA.2016.30
[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Jun 2016). https://doi.org/10.1109/cvpr.2016.90
[15] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W. Fletcher. 2021. Mind Mappings: Enabling Efficient Algorithm-Accelerator Mapping Space Search *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 943–958. https://doi.org/10.1145/3445814.3446762
[16] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraş. 2019. Terminal Brain Damage: Exposing the Graceless Degradation

in Deep Neural Networks under Hardware Fault Attacks. In *Proceedings of the 28th USENIX Conference on Security Symposium* (Santa Clara, CA, USA) *(SEC'19)*. USENIX Association, USA, 497–514.

[17] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. 2020. DeepSniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 385–399. https://doi.org/10.1145/3373376.3378460

[18] 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)*. Association for Computing Machinery, New York, NY, USA, 349–354. https://doi.org/10.1145/3489517.3530439

[19] 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)*. Association for Computing Machinery, New York, NY, USA, 726–741. https://doi.org/10.1145/3470496.3527418

[20] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah, J. Demmel, J. Wawrzynek, and Y. Shao. 2021. CoSA: Scheduling by Constrained Optimization for Spatial Accelerators. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, Los Alamitos, CA, USA, 554–566. https://doi.org/10.1109/ISCA52012.2021.00050

[21] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167 [cs.LG]

[22] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[23] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 1651–1669. https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[24] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. 2022. ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1237–1254. https://doi.org/10.1109/MICRO56248.2022.00086

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[26] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An Open Framework for Architecting Trusted Execution Environments. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 38, 16 pages. https://doi.org/10.1145/3342195.3387532

[27] 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)*. 229–243. https://doi.org/10.1109/HPCA53966.2022.00025

[28] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science* (2016), 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

[29] Sanu K. Mathew, Farhana Sheikh, Michael Kounavis, Shay Gueron, Amit Agarwal, Steven K. Hsu, Himanshu Kaul, Mark A. Anders, and Ram K. Krishnamurthy. 2011. 53 Gbps Native GF$(2^4)^2$ Composite-Field AES-Encrypt/Decrypt Accelerator for Content-Protection in 45 nm High-Performance Microprocessors. *IEEE Journal of Solid-State Circuits* 46, 4 (2011), 767–776. https://doi.org/10.1109/JSSC.2011.2108131

[30] Bert Moons and Marian Verhelst. 2017. An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS. *IEEE Journal of Solid-State Circuits* 52, 4 (2017), 903–914. https://doi.org/10.1109/JSSC.2016.2636225

[31] Onur Mutlu and Jeremie S. Kim. 2020. RowHammer: A Retrospective. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 39, 8 (Aug. 2020), 1555–1571. https://doi.org/10.1109/TCAD.2019.2915318

[32] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[33] Beomsik Park, Ranggi Hwang, Dongho Yoon, Yoonhyuk Choi, and Minsoo Rhu. 2022. DiVa: An Accelerator for Differentially Private Machine Learning. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1200–1217.

[34] Michael Pellauer, Yakun Sophia Shao, Jason Clemons, Neal Crago, Kartik Hegde, Rangharajan Venkatesan, Stephen W. Keckler, Christopher W. Fletcher, and Joel Emer. 2019. Buffets: An Efficient and Composable Storage Idiom for Explicit Decoupled Data Orchestration. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 137–151. https://doi.org/10.1145/3297858.3304025

[35] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T. Lee, Hsien-Hsin S. Lee, Gu-Yeon Wei, and David Brooks. 2021. Cheetah: Optimizing and Accelerating Homomorphic Encryption for Private Inference. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 26–39. https://doi.org/10.1109/HPCA51647.2021.00013

[36] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640 [cs.CV]

[37] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 2007. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. 183–196. https://doi.org/10.1109/MICRO.2007.16

[38] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. arXiv:2112.10752 [cs.CV]

[39] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose A. Joao, and Moinuddin K. Qureshi. 2018. Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 416–427. https://doi.org/10.1109/MICRO.2018.00041

[40] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 238–252.

[41] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381 [cs.CV]

[42] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. 2001. A Compact Rijndael Hardware Architecture with S-Box Optimization. In *Advances in Cryptology — ASIACRYPT 2001*, Colin Boyd (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 239–254.

[43] Arne Symons, Linyan Mei, Steven Colleman, Pouya Houshmand, Sebastian Karl, and Marian Verhelst. 2022. Towards Heterogeneous Multi-core Accelerators Exploiting Fine-grained Scheduling of Layer-Fused Deep Neural Networks. arXiv:2212.10612 [cs.AR]

[44] László Szekeres, Mathias Payer, Tao Wei, and Dawn Song. 2013. SoK: Eternal War in Memory. In *2013 IEEE Symposium on Security and Privacy*. 48–62. https://doi.org/10.1109/SP.2013.13

[45] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. 2018. VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (Williamsburg, VA, USA) *(ASPLOS '18)*. Association for Computing Machinery, New York, NY, USA, 665–678. https://doi.org/10.1145/3173162.3177155

[46] Florian Tramer and Dan Boneh. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJVorjCcKQ

[47] 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)*. USENIX Association, Carlsbad, CA, 681–696. https://www.usenix.org/conference/osdi18/presentation/volos

[48] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52 (04 2009), 65–76. https://doi.org/10.1145/1498765.1498785
[49] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. 2019. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942149
[50] Wenjie Xiong, Liu Ke, Dimitrije Jankov, Michael Kounavis, Xiaochen Wang, Eric Northup, Jie Amy Yang, Bilge Acun, Carole-Jean Wu, Ping Tak Peter Tang, G. Edward Suh, Xuan Zhang, and Hsien-Hsin S. Lee. 2022. SecNDP: Secure Near-Data Processing with Untrusted Memory. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 244–258. https://doi.org/10.1109/HPCA53966.2022.00026
[51] Chenyu Yan, D. Englender, M. Prvulovic, B. Rogers, and Yan Solihin. 2006. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *33rd International Symposium on Computer Architecture (ISCA'06)*. 179–190. https://doi.org/10.1109/ISCA.2006.22
[52] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. 2020. DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1463–1480. https://www.usenix.org/conference/usenixsecurity20/presentation/yao
[53] Yiqun Zhang, Kaiyuan Yang, Mehdi Saligane, David Blaauw, and Dennis Sylvester. 2016. A compact 446 Gbps/W AES accelerator for mobile SoC and IoT in 40nm. In *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*. 1–2. https://doi.org/10.1109/VLSIC.2016.7573553

A ARTIFACT APPENDIX

A.1 Abstract

Our artifact provides the source code of SecureLoop, the template/example architecture and workload descriptions, and other utility functions. We provide the top-level testbench as a Jupyter notebook (`workspace/run_all.ipynb`) and a script (`workspace/scripts/fig11.sh`) that runs all three steps of our scheduling algorithm to generate the stats (latency, energy, off-chip traffic) for secure DNN accelerators. Running the notebook reproduces the results in Figure 11 of this paper. We use a docker environment to manage all dependencies necessary to run our artifact. Our artifact requires a x86-64 machine and 15GB of disk space for docker support.

A.2 Artifact check-list (meta-information)

- **Algorithm**: Scheduling of secure DNN accelerators with cryptographic engines
- **Program**: Python3
- **Run-time environment**: Dockerfile
- **Hardware**: x86-64 machine
- **Output**: Plots and stats (csv) generated from the Jupyter notebook
- **Experiments**: Comparison of different scheduling algorithms (latency, additional off-chip traffic due to cryptographic operations) for various DNN workloads
- **How much disk space required (approximately)?**: 15GB
- **How much time is needed to prepare workflow (approximately)?**: 30 minutes if pulling docker image using the provided `docker-compose.yaml.template` file. 2 hours if building docker images from the sources.
- **How much time is needed to complete experiments (approximately)?**: 3 hours for running all three DNN workloads (AlexNet, ResNet18, MobilenetV2) on a default DNN accelerator architecture setup
- **Publicly available?**: Yes, available at https://github.com/kyungmi-lee/SecureLoop-MICRO2023Artifact
- **Code licenses (if publicly available)?**: MIT
- **Archived (provide DOI)?**: 10.5281/zenodo.8329657

A.3 Description

A.3.1 How to access. The artifact including the source code for SecureLoop, the Jupyter notebooks, and the scripts that run experiments is available at https://github.com/kyungmi-lee/SecureLoop-MICRO2023Artifact.

A.4 Installation

We provide a docker image that provides the necessary infrastructure. The installation process involves installing a docker app, then pulling a docker image using the provided `docker-compose.yaml.template` file. We also provide an option to build docker images using the sources instead of pulling the pre-built image. Please check `README.md` with the artifact repository for installation and setup.

A.5 Experiment workflow

The experiment workflow is outlined in the Jupyter notebook `workspace/run_all.ipynb`. First, the DNN accelerator / cryptographic engine architecture and a DNN workload are defined. Then, the notebook goes through all three steps in our scheduling algorithm (loopnest scheduling, authentication block assignment, and simulated annealing for joint-layer search). Finally, it generates plots in Figure 11 comparing different scheduling algorithms.

Alternatively, a user can run a script `workspace/scripts/fig11.sh` in a terminal, and the necessary scheduling and evaluation codes are executed for three workloads in Figure 11. The plots can be generated by a shorter Jupyter notebook `workspace/plot_figures.ipynb` once the script is finished.

A.6 Evaluation and expected results

For each DNN workload, the notebook generates two plots to compare different scheduling algorithms: 1) performance overhead in the nomalized latency (Figure 11(a)), and 2) additional off-chip traffic due to cryptographic operations (Figure 11(b)). Different DNN workloads can be chosen by commenting in/out workload definitions in the notebook. The generated plots for each workload should match those in Figure 11. However, note that the scheduling algorithm involves random processes (e.g., simulated annealing randomly chooses which layer and loopnest schedule to use at each iteration), and the result might not exactly match the numbers in Figure 11. Nevertheless, the result should be close (e.g., for MobilenetV2, performance overhead for *Crypt-Opt-Cross* can vary between 9.70 to 9.99, while the number in Figure 11 is 9.86; for AlexNet and ResNet18, the results only deviate by < 0.01), and the general trend between different scheduling algorithms should be the same.

A.7 Experiment customization

The DNN accelerator / cryptographic engine architecture in the notebook can be modified to run design space exploration experiments. Running the scheduling algorithm as detailed in the notebook also generates raw data and a csv file summarizing the stats inside the folder `workspace/designs/{design_name}/{design_version}`.
We provide a python script to generate architecture configurations

(`workspace/generate_arch.py`). Also, an additional script `workspace/scripts/fig14.sh` illustrates how to configure the python scripts to evaluate architectures with different PE array shapes (Figure 14).

A.8 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html